Fitting Model Parameters to Patient Data Automatically Through Evolutionary Computation

Michael Wesley Sagan
Marquette University

FITTING MODEL PARAMETERS TO PATIENT DATA AUTOMATICALLY THROUGH EVOLUTIONARY COMPUTATION

by

Michael W. Sagan, B.S.

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

May 2011

ABSTRACT
FITTING MODEL PARAMETERS TO PATIENT DATA AUTOMATICALLY THROUGH EVOLUTIONARY COMPUTATION

Michael W. Sagan, B.S.

Marquette University, 2011

The Orthopaedic and Rehabilitation Engineering Center (O.R.E.C) has been studying two conditions which can wreak havoc on the human body's balance control system. The conditions being studied are cerebral palsy (CP) and adolescent idiopathic scoliosis (AIS)[1, 2, 3, 4, 5, 6].

A greater understanding of postural deficits in children with these conditions and the causes for these conditions may help researchers achieve better interventions and treatments [3]. A tool used to characterize postural stability is a bi-planar postural stability model. This model is designed where 10 parameter values have to be adjusted until the simulated data is statistically the same as the actual patient data. This process becomes a minimization problem in a 10-dimensional space.

A previous method for input parameter value adjustment involves an operator manually adjusting each parameter value. This process is very time consuming and requires approximately 4 uninterrupted days for computation and analysis. An improved automatic method brought the computation and analysis time down to a reasonable 12 hours [1]. To further improve the input process for parameter values new search methods were explored, including evolutionary algorithms.

A evolutionary algorithm was integrated with the existing bi-planar postural stability model [3, 4, 5, 6, 1, 2] to produce more accurate results more efficiently by exploring the full ten-dimensional search space using a previously-designed cost function [1, 2].

Simulation time was drastically improved after streamlining the existing code provided by Andrew Sovol [1, 2]. After making improvements, a single iteration ran 13.21 times faster on average. The average time for a simulation was reduced to 6005.74 seconds (approximately 1 hour and 40 minutes) from 12 hours. Statistical mismatches were also computed with a smaller p-value reducing the possible error from 10% to 0.5%.

ACKNOWLEDGMENTS

Michael W. Sagan, B.S.

I wish to thank those that have helped me during my graduate career and along the way to a completed thesis:

- Professor Susan Riedel for her invaluable guidance during both my graduate career as a advisor and as a teacher during my undergraduate years.
- Dr. Gerald Harris and Dr. Ronald Brown for being on my committee and for their useful feedback in revising the thesis.
- Dr. Karla Bustamante for her creation of the bi-planar stability model, as it gave an interesting platform in which to utilize an evolutionary algorithm.
- Andrew Sovol for the improvements to the model, the construction of the cost function, and his assistance during the early stages of algorithm development.
- Orthopaedic and Rehabilitation Engineering Center at Marquette University for the use of the model and data sets.
- Father Louis, mother Holly, brother Peter, and sister Amanda for their support.
- Fiancée Emily, I could not have done this without your unwavering support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Statement

Am evolutionary algorithm can be integrated with an existing bi-planar postural stability model [3, 4, 5, 6, 1, 2] to produce more accurate results more efficiently by exploring the full ten-dimensional search space using a previously-designed cost function [1, 2].

1.2 Background

Two-thirds of human body mass is located two-thirds of body height above the ground. Because of this, a human standing upright is an inherently unstable system, unless a control system is continuously acting. The degeneration of the balance control system in many pathologies has forced researchers to understand more about the system and to quantify its status at any point in time[9].

The Orthopaedic and Rehabilitation Engineering Center (O.R.E.C) has been studying two conditions which can wreak havoc on the body's balance control system. The conditions being studied are cerebral palsy (CP) and adolescent idiopathic scoliosis (AIS)[1, 2, 3, 4, 5, 6].

1.2.1 Cerebral Palsy

By definition, the impairment known as cerebral palsy (CP) describes damage to the immature brain resulting in problems with balance, coordination, and movement [10]. In 2000, the prevalence of CP was an estimated 3.1 per 1,000 (or about 1 in 323) 8-year-olds [11].

There are many possible causes for CP. Some of these causes are in brain development during the first 6 months of pregnancy. Theses causes include genetic conditions and limited blood supply to the brain [10]. Other causes of CP happen after the brain has developed, such as bacterial meningitis and other infections, bleeding in the brain, lack of oxygen, severe jaundice, and head injury [10].

Cerebral palsy is a permanent condition that can cause both muscle tightness and uncontrolled reflex muscle movements [1]. These conditions can impair postural stability during quiet standing, and are often treated with physical therapy and other assistive measures. It is very important to track the success of the assistive measures in order to evolve programs for an individual patient and others.

The postural control model developed by Bustamante [3, 4, 5, 6, 2] has shown differences between typical children and children with CP. This model also allowed researchers to look at differences between different sub-populations of children with CP. The model also allows researchers to distinguish differences between patients that have gone through physical therapy and different assistive measures to determine the success of these interventions.

1.2.2 Adolescent Idiopathic Scoliosis

Scoliosis is an abnormal curvature of the spine that affects approximately seven million people in the United States [12]. It is estimated that 3 to 5 out of every 1,000 children develop spinal curvatures that are large enough to require treatment [13].

AIS can affect the ability of an individual to establish and continue to maintain balance during quiet standing [6]. If the spinal deformity cannot be controlled by non-surgical assistive measures then surgical intervention may be the next option. One surgical method for controlling AIS is posterior spinal fusion (PSF) [6].

It is very important to have metrics for determining the success of the surgery. The postural control model developed by Bustamante [3, 4, 5, 6, 2] has successfully distinguished a group of patients with AIS from a control group of typical adolescents, and shows differences between AIS patients before and after PSF surgery.

1.3 Force Plate Testing

In order to test for postural stability, researchers must know the patient's center of gravity (COG) and center of mass (COM). The center of gravity is the vertical vector from the center of mass intersecting the horizontal plane. Balance assessment is normally measured by tracking the patient's center of pressure. The center of pressure (COP) is directly linked to the center of gravity because the center of pressure is the neuromuscular reaction needed to control the center of gravity locus [3].

The system implemented to find the center of pressure in each plane consisted of two force plates (AMTI ORS6-500). These plates acquire forces (F) and moments (M) in the x, y, and z directions. The force plate outputs are amplified using the AMTI amplification unit and are sampled at 100 Hz using a National Instruments Data Acquisition Card (PCI-6034-E) [3].

The equations used to find the center of pressure (COP) coordinates in both the anteroposterior and mediolateral planes, where k is used to combine

the data from 2 force plates into a single ML COP value, are as follows:

$$COP_{AP} = \frac{M_y + F_x \times Z_0}{F_z} - Y_0 \tag{1.1}$$

$$COP_{ML} = \frac{M_x - F_y \times Z_0}{F_z} + X_0 \tag{1.2}$$

X_0, Y_0, and Z_0 represent the offset of the estimated center of the force plate to the top surface of the force plate. Once Equation 1.1 and Equation 1.2 have been calculated for each force plate, the resultant AP and ML values can be calculated with the following equations [4]:

$$COP_{R(ML)} = \frac{(COP_{ML(right)} + k) \times F_{z(right)} + (COP_{ML(left)} - k) \times F_{z(left)}}{F_{z(left)} + F_{z(right)}} \tag{1.3}$$

$$COP_{R(AP)} = \frac{COP_{AP(right)} \times F_{z(right)} + COP_{AP(left)} \times F_{z(left)}}{F_{z(left)} + F_{z(right)}} \tag{1.4}$$

The center of pressure data is used by researchers to calculate sway metrics, which are shown to be an accurate way to characterize the balance control system [2]. The sway metrics used by researchers can be found in Table 1.1. Thirteen sway metrics are calculated based upon the above COP equations. Equations 1.5 through 1.20 are used to calculate the sway metric values. In these equations n is the number of samples (2000), T is the testing time (20 seconds), $P(f)$ is the power spectrum, Δf is the frequency increment within the time domain signal [1].

Metric	Units
Mean Distance (MD)	mm
Root Mean Squared (RMS)	mm
Range (Range)	mm
Total Traveled Distance (TX)	mm
Mean Velocity (MV)	$\frac{mm}{s}$
Mean Frequency (MF)	Hz
Sway Area (SA)	$\frac{mm^2}{s}$
Total Power (TP)	mm^2
Centroidal Frequency (CP)	Hz
Frequency Dispersion (FD)	-
Median Power Frequency (FP50)	Hz
95% Power Frequency (FP95)	Hz

Table 1.1: Sway Metrics

$$MD = \frac{\sum |COP|}{n} \tag{1.5}$$

$$RMS = \sqrt{\frac{\sum |COP|^2}{n}} \tag{1.6}$$

$$Range = MAX(COP) - MIN(COP) \tag{1.7}$$

$$TX = \sum_{i=1}^{n-1} \{[COP_{(ap_{i+1})} - COP_{(ap_i)}]^2 + [COP_{(ml_{i+1})} - COP_{(ml_i)}]^2\} \tag{1.8}$$

$$MV = \frac{TX}{T} \tag{1.9}$$

$$MF = \frac{MV}{4\sqrt{2}MD} \tag{1.10}$$

$$SA = \frac{1}{2T}\sum_{i=1}^{n-1}|COP_{(ap_{i+1})} \times COP_{(ml_i)} - COP_{(ap_i)} - COP_{(ml_{i+1})}| \tag{1.11}$$

$$RD = \frac{1}{n}\sum_{i=1}^{n-1}\sqrt{[COP_{(ap)} - \frac{\sum COP_{ap}}{n}]^2 + [COP_{(ml)} - \frac{\sum COP_{ml}}{n}]^2} \tag{1.12}$$

$$TP = \sum_{n_0}^{n_1} P(f) \tag{1.13}$$

$$n_0 = \frac{0.15}{\Delta f} \tag{1.14}$$

$$n_1 = \frac{5}{\Delta f} \tag{1.15}$$

$$CF = \sqrt{\frac{\mu_2}{\mu_0}} \tag{1.16}$$

$$\mu_k = \sum_{m=n_0}^{n_1} (m \times \Delta f)^k P(m) \tag{1.17}$$

$$FD = \sqrt{1 - \frac{{\mu_1}^2}{\mu_0 \mu_2}} \tag{1.18}$$

$$FD50 = \sum_{i=n_0}^{n_1} P(f) \geq 0.50 \times TP \tag{1.19}$$

$$FD95 = \sum_{i=n_0}^{n_1} P(f) \geq 0.95 \times TP \tag{1.20}$$

Chapter 2

STATEMENT OF PROBLEM

2.1 Existing Model

Many researchers use simulations in order to gain a better understanding of a particular physical system. A better understanding of postural deficits in children with various conditions and the causes for these conditions may help researchers achieve better interventions and treatments [3].

In order to characterize postural stability in the anteroposterior plane Maurer et al.[14] implemented a simplified proportional-integral-derivative (PID) model. Bustamante et al. used the same model and applied it to produce accurate results in the mediolateral plane. Combining these two models yields a bi-planar model for postural stability. A block diagram of this model is shown in Fig. 2.1. The model requires the parameters found in Table 2.1 for both the anteroposterior and mediolateral plane, totaling 10 parameters.

These 10 parameters have to be adjusted until the simulated data created by the model is statistically the same as the actual patient data. This process becomes a minimization problem in a 10-dimensional space.

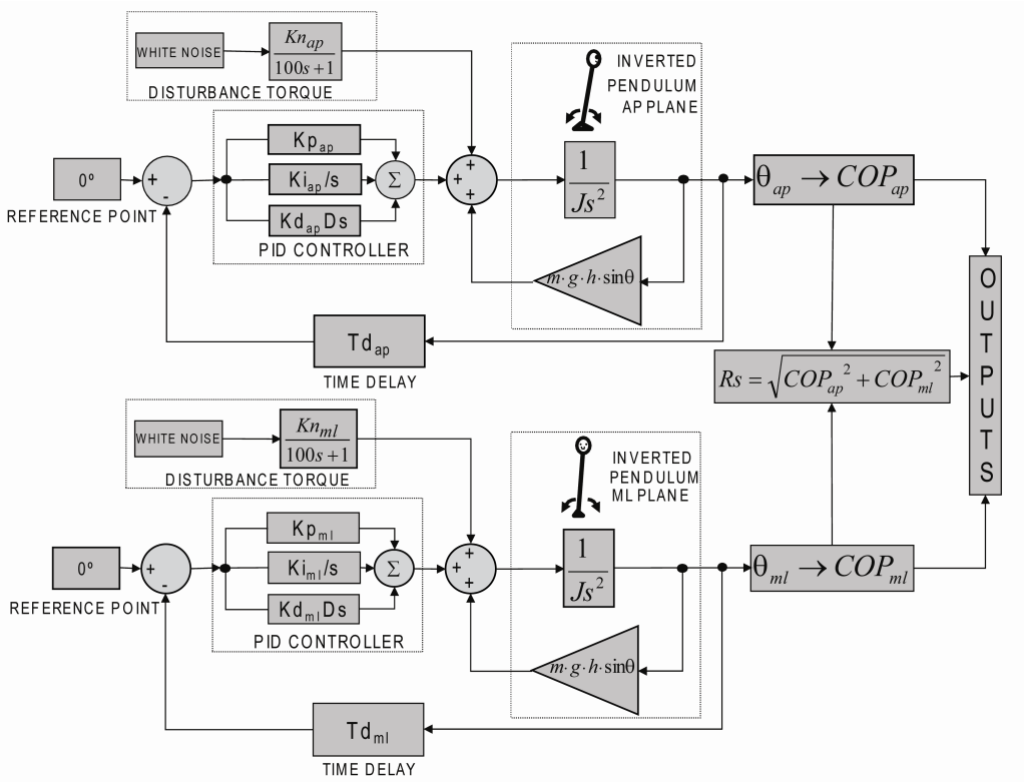

Figure 2.1: Bi-Planar Model for Postural Stability Block Diagram [3]

Parameter	Definition	Units
Kp	Proportional Gain	$\frac{Nm}{deg}$
Ki	Integral Gain	$\frac{Nm}{s \cdot deg}$
Kd	Derivative Gain	$\frac{Nm \cdot s}{deg}$
Td	Time Delay	s
Kn	Noise Gain	$const.$

Table 2.1: Model Parameters

2.2 Manual Simulation Operation

2.2.1 Overview

The manual operation of this simulation is a time consuming process. The operator must complete one iteration of this simulation at a time, carefully selecting values for the 10 input parameters and monitoring the output of the model [1]. The results from the simulation are presented to the operator at the command prompt, and it is the responsibility of the operator to record

the past inputs and statistical mismatches. At the completion of a single iteration, the operator will then change a parameter in the model in order to try and minimize the number of statistical mismatches in the next iteration.

2.2.2 Limitations

The main limitation of the manual simulation method is the amount of time it takes the operator to fit parameters to the patient data. Each iteration for the operator takes approximately 20 minutes [1]. The total computation and analysis time for a manual simulation is approximately 4 uninterrupted days. A typical fit of the model to patient data takes at least 2 weeks [1].

2.3 Automatic Iterative Simulation Operation

2.3.1 Overview

The automatic iterative simulation process was a great improvement over the manual simulation process. The time required to complete a single iteration of the model was reduced to 2 minutes and 45 seconds from 20 minutes using the rapid accelerator package from Simulink. The improvement of the iteration time brought the computation and analysis time down to a reasonable 12 hours [1] with a typical pace of about a single day.

2.3.2 Limitations

There were a multitude of limitations with the final version of the automatic parameter fitting method. One of the largest problems was the excessive amount of plots that were drawn by MATLAB for every iteration. Each iteration involves the plotting of 74 individual windows. During this plotting time, extraneous information is printed to the command window which is not needed by the operator. The large number of plot windows requires a large amount of computer

resources which makes it difficult to run on older computer hardware. This method also required that the user supply a starting point for the algorithm. There are many instances where a particular choice of parameter values creates errors in the simulation and the operator has to restart the application.

The automatic parameter fitting technique limited the parameter values to $\pm 10\%$ on either side of the starting parameter value. If the set of parameter values that minimize the cost function of the automatic parameter fitting technique contains a single parameter that is more than 10% away from the starting parameter value the optimal solution will not be found. This limitation places a great deal of importance on the initial parameter values for the simulation. It is difficult to determine the proper starting parameters for an unknown patient data set. A method using the entire search space is more likely to find model parameters that minimize the statistical number of mismatches between model and patient data without a required user input.

Another limitation of this method is the level of significance given for the t-test. Previous trials [1, 2, 5] used an unpaired t-test with a p value of 0.1 and stated that there were no significant differences. With a level of significance of 0.1, there is a 10% chance of rejecting the null hypothesis when the null is true and conclude there is a group difference when there really is no group difference at all. A smaller p value such as 0.005 would make this error possibility 0.5% versus 10%.

Chapter 3

EVOLUTIONARY COMPUTING

3.1 Evolution

Evolution (L. *evolvere*, to unfold) is defined as "genetic change in a population of organisms; in general, evolution leads to progressive change from simple to complex"[15]. Charles Darwin was the first to propose natural selection as an explanation for the mechanism of evolution that produced the diversity of life on earth. His hypothesis grew from his observations on a five-year voyage around the world aboard the H.M.S. *Beagle* [15].

Natural selection, the driving force of evolution, was discovered through the observations of finches on the Galápagos Islands, 540 miles off the cost of Ecuador. Darwin encountered 14 species of finches on various islands. He saw that they were similar, but varied slightly in appearance, mainly the shape of the beaks. Darwin thought it was reasonable to assume there was one common ancestor and the birds had adapted through millions of years to different beaks due to different diets on each island.

Darwin wrote:

> "Can we doubt... that individuals having any advantage, however slight, over others, would have the best chance of surviving and procreating their kind? On the other hand, we may feel sure that any variation in the least degree injurious would be rigidly destroyed.

> This preservation of favorable variations, I call Natural Selection [15]."

In a given population, those individuals that possess superior physical, behavioral, or other traits are more likely to survive. Through survival, these individuals are able to pass on their favorable characteristics to their offspring.

Over time, the entire population of the ecosystem is said to evolve to contain organisms that, on average, are more fit than those of previous generations of the population because they exhibit more of those characteristics that tend to promote survival [16]. The driving force behind evolution, natural selection, is also commonly referred to as "survival of the fittest."

3.2 Evolutionary Computation

Evolutionary computation (EC) uses evolutionary principles to construct algorithms that can be used to find optimal solutions for a given problem. Traditional search algorithms can be used for a search space with only a small number of possible solutions. In these situations all the solutions can be examined in a reasonable amount of time (exhaustive search) and the optimal solution will be found [16]. The use of exhaustive search becomes impractical as the search space grows in size.

The key difference between evolutionary algorithms (EA) and traditional search algorithms is that evolutionary algorithms are based upon a population. A population, just as in nature, is a set of individuals. The individuals are possible solutions to the problem. Every individual has a fitness value which is the difference between a solution value and the optimal solution of the problem. The metaphor between problem solving and evolution can be found below in Table 3.1.

Evolution		**Problem Solving**
Environment	$\longleftrightarrow$	Problem
Individual	$\longleftrightarrow$	Candidate Solution
Fitness	$\longleftrightarrow$	Solution Quality

Table 3.1: Evolutionary Computing Metaphor [7]

3.3 Algorithm Overview

All evolutionary algorithms consist of a common framework. In order to create a complete algorithm it is necessary to specify some common components. The necessary components for any evolutionary algorithm consist of the solution representation, population initialization, population size, parent selection, recombination, mutation, fitness function, survivor selection, and termination criteria.

3.3.1 Solution Representation

In nature a given trait of an individual (phenotype) is represented by a genetic code (genotype). In evolutionary computing it is no different. Individuals can be represented in a number of different ways. The mapping of a given characteristic or solution can use binary strings, integers, or floating point values. Binary strings are used as a geneotypic representation of a real value, and are primarily used in Genetic Algorithms (GA). Evolutionary Programming (EP) and Evolutionary Strategies (ES) use real value vectors to represent individuals in the population. Many times these representations do not have a genotypic representation, only a phenotypic one.

3.3.2 Population Initialization

At the beginning of each evolutionary algorithm a population needs to be initialized before the evolutionary process can be started [17]. This initialization can be either random or using problem-specific heuristics. This initialization is

based on the known search space of a problem, and is directly related to the population size.

3.3.3 Population Size

The population size of an evolutionary algorithm is the number of candidate solutions available. This size is particularly critical in a generational model where the entire population is being replaced. The larger the population size, the more potential search space can be explored in a single generation.

3.3.4 Parent Selection

The role of parent selection, otherwise known as mating selection, is to distinguish among individuals based on their quality (fitness) [7]. This process allows better individuals to become parents of the next generation. Parent selection, along with survivor selection, is the driving mechanism for improving the quality of the entire population.

3.3.5 Recombination

The principle behind recombination is simple: by mating individuals with different but desirable traits an offspring can be created that combines those traits [7]. The classic two-parent reproductive mechanism is recombination in which subcomponents of the parents' genomes are cloned and reassembled to create an offspring genome [18]. Simple fixed-length genomes traditionally use "crossover" operators. In this case, a crossover point marks the point where the parents' genome would be split and reassembled. An example of one-point crossover is shown below in Figure 3.1.

$$\begin{array}{rcl} Parent1 = [A\ B\ C\ D\ E \mid F\ G\ H\ I\ J] & \searrow & Child1 = [A\ B\ C\ D\ E \mid U\ T\ S\ R\ Q] \\ Parent2 = [Z\ Y\ X\ W\ V \mid U\ T\ S\ R\ Q] & \nearrow & Child2 = [Z\ Y\ X\ W\ V \mid F\ G\ H\ I\ J] \end{array}$$

Figure 3.1: One-Point Crossover

3.3.6 Mutation

Crossover exploits the current solution to find better ones, whereas mutation explores the entire search space [16]. Mutation is the method that attempts to maintain genetic diversity in the population. The probability of mutation is generally $\frac{1}{L}$, where L is the length of the individual string.

3.3.7 Fitness Function

The quality of an individual is referred to as the fitness. This fitness value is used in an EA to determine which individuals should be selected to survive and reproduce [19]. The evaluation metric for EAs is called the fitness function. The fitness function is designed by the algorithm designer to give continuous feedback on how well an individual candidate solution functions in the environment.

3.3.8 Survivor Selection

Survivor selection is the method used by the algorithm to filter out undesirable individuals from the population. This selection process can also be viewed as population replacement. The selection method is often deterministic and based on two methods [7].

One method of survivor selection is fitness based, which involves ranking the parents and the offspring and taking the best individuals from that group. This method is considered to be elitist. Any genetic algorithm is said to be elitist if the following is true [20]:

Let k be the best individual at time t. At $t + 1$ either:

1. k is in the population, or

2. Something better than k is in the population

The advantage of elitism is that high fitness solutions are not lost in the next generation. The second method is age based, which has the algorithm make as many offspring as there were parents, and the entire population is replaced. This method does not keep the high fitness solutions from previous generations.

3.3.9 Termination Criteria

There are two ways to determine a stopping condition for an EA. If a known stopping solution (a certain fitness) is desired then the algorithm can be stopped when the fitness of a candidate solution has met or surpassed that fitness goal. But for some EAs, a solution may not be achievable or known. If the fitness goal cannot be met or it is not known when the best solution would arrive, there are 4 common ways of building termination conditions [7].

1. Maximum CPU time elapses;
2. Total number of fitness evaluations reaches a given time limit;
3. The fitness improvement remains under a threshold value for a given period of time;
4. The population diversity drops under a given threshold.

After a termination criterion has been met, the best solution found up until that termination time is returned to the user.

Chapter 4

METHODS

4.1 Introduction

This chapter covers the specifics of the algorithm framework discussed in Chapter 3. Differences between the previous Iterative method and the new Evolutionary method are discussed in this chapter. Instructions for simulation installation and operation are covered in section 4.6.

4.2 Code Clean Up

Implementation of an evolutionary algorithm into the simulation of the bi-planar model of postural stability required the original code to be revised. Every file used in the simulation was reviewed line by line. All lines which did not have a semi-colon suppressor were fixed in order to remove that output from the command window. All information which would be useful to the operator was explicitly set using 'fprintf' and 'display' statements. In addition to the console outputs, all unnecessary plotting functions were removed from the program in order to speed the simulation up.

4.3 Overview of Simulation Improvements

The implementation of an evolutionary algorithm and a higher accuracy t-test has been integrated into graphical user interfaces (GUIs) previously developed by Andrew Sovol [1, 2] shown in Figure 4.1. The changes in the GUI are listed in section 4.6.1 below.

Figure 4.1: Original GUI [1, 2] vs. New GUI

The iterative method created by Andrew Sovol [1, 2] continues to require the same inputs needed in the previous software release. The new algorithm can be given starting values or default values can be used. If a user defines starting parameter values, the algorithm will build a search space that is ± 50% of the starting value. If no parameters are defined, a search space that is ± 75% of the "Normal" values is created. The values for the "Normal" parameters can be found in Table 4.1.

	Eyes Open		Visual Feedback		Eyes Closed	
	AP	ML	AP	ML	AP	ML
Kp	16.7	16.7	17.5	17.1	17.55	16.5
Ki	0.6	0.6	0.6	2.0	0.8	1.0
Kd	6.7	9.0	6.7	9.0	6.5	8.9
Td	0.171	0.171	0.171	0.171	0.171	0.171
Kn	250	118	245	113	327	130

Table 4.1: Normal Adult Parameters [8]

4.4 Algorithm Overview

This section describes the framework of the implemented evolutionary algorithm. The components for the evolutionary algorithm consist of the solution representation, population initialization, population size, parent selection, recombination, mutation, fitness function, survivor selection, and termination criteria.

4.4.1 Solution Representation

During the course of building this algorithm, two types of solution representation were used. Initially a binary representation was used, however it was ultimately decided to use floating point values in order to remove the need for a genotypic representation. The individual floating point numbers are the exact inputs to the simulation and are manipulated directly. An example of a single individual can be found in Fig. 4.2.

$$[\ kpa \quad kia \quad kda \quad tda \quad kna \quad kpm \quad kim \quad kdm \quad tdm \quad knm \quad fitness\ ]$$

Figure 4.2: Individual Population Member

4.4.2 Population Initialization

The population is initialized using a pseudo-random number generator built into MATLAB. The random value generator is seeded with the current clock time. The “rand” function generates a random floating point value between

0 and 1. The arguments for the "rand" function determine the size of the output matrix. Figure 4.3 shows the MATLAB statement that initializes the population. The "rand" function builds a matrix of random numbers with "populationsize" number rows in a single column. These random values are multiplied by the difference between the upper and lower bounds of the parameter's search space. This value is added to the lower bound for each parameter. These bounds are based on ±75% of "normal" adult values. During the initialization period all fitness values are set to a maximum value of 999. Each of the parameters is represented by a matrix with "populationsize" number of rows and one column. These matricies are then combined to make the "population" matrix which is "populationsize" rows by 11 columns.

```
1  population=[kpamin+(rand(populationsize,1)*kpadiff),...
2  kiamin+(rand(populationsize,1)*kiadiff),...
3  kdamin+(rand(populationsize,1)*kdadiff),...
4  tdamin+(rand(populationsize,1)*tdadiff),...
5  knamin+(rand(populationsize,1)*knadiff),...
6  kpmmin+(rand(populationsize,1)*kpmdiff),...
7  kimmin+(rand(populationsize,1)*kimdiff),...
8  kdmmin+(rand(populationsize,1)*kdmdiff),...
9  tdmmin+(rand(populationsize,1)*tdmdiff),...
10 knmmin+(rand(populationsize,1)*knmdiff),fitness]
```

Figure 4.3: Initialization Code

4.4.3 Population Size

The population size is specified by the operator at simulation time. However, the population must be larger than 10 individuals. This value is determined by the number of possible solutions the operator would like to try before any "selecting" is performed by the algorithm. In the tests conducted for this study, a population of 100 members was used.

If during the evaluation of the initial population members there are less than 2 viable parents, the algorithm will fail because there are no parents to mate. In order to explore a large search space, a larger initial population

should be used. The algorithm is performing a random search of the search space using the random values initialized in the population matrix.

4.4.4 Parent Selection

The population is ranked based upon fitness, with the solutions having the fewest statistical mismatches at the top of the population. Parents are selected randomly using the "randi" that randomly generates an integer between the values of 1 and 10. This method is inherently greedy as it is only choosing from the best individuals. After the parents are chosen recombination can begin.

4.4.5 Recombination

Recombination is performed using a one-point crossover as described in Chapter 3 (See Fig. 3.1). One-point crossover is generally used in binary representations [7], however it was a good choice for this representation as well. Using the solution representation shown in Fig. 4.2 we can build children using the MATLAB code shown in Fig. 4.4:

```
1 child1=[(population(parents(1),1:5)),(population(parents(2),6:10)),0];
2 child2=[(population(parents(2),1:5)),(population(parents(1),6:10)),0];
```

Figure 4.4: Building Two Children from Two Parents

This code takes the values 1-5 of the first parent and the 6-10 values of the second parent and constructs the first child with a placeholder for the fitness. The second line performs a similar operation to create the second child. One-point crossover can be performed on this representation because the order of magnitudes of the individual parameters are the same for the AP and ML plane.

4.4.6 Mutation

Due to the varying magnitudes of each parameter, ranging from 0.171 for Td in a normal adult to 327 for Kn, a different mutation approach had to be used. Many algorithms use a single step size, or a varying step size based on the parameter to be mutated. There is a large range in the values between these individual parameters, therefore a single step size was not practical. If a single step size was used, and for instance was chosen to be 1, the step size for Td would be much too large and would be much too small for Kn.

Instead of using a given step size, a scaling factor was used as mutation. Figure 4.5 lists the MATLAB code that constructs a mutant allele. The algorithm randomly selects a characteristic to mutate. Four parameters are chosen to mutate using a scalar factor that is a random value between 0 and 1 or between 1 and 2. This scaling mutation removes the need to adjust the step size for a single parameter.

```
1 mutantcharacterisitc=randi([1,10]);
2 mutantvalue1=child1(mutantcharacterisitc)*rand;
3 mutantvalue2=child2(mutantcharacterisitc)*rand;
4 mutantvalue3=child1(mutantcharacterisitc)*(1+rand);
5 mutantvalue4=child2(mutantcharacterisitc)*(1+rand);
```

Figure 4.5: Construction a Mutant Allele

4.4.7 Fitness

The fitness function of this algorithm is directly tied to the actual output of the model. Andrew Sovol [1, 2] constructed a cost function for his iterative minimization method. This cost function was the sum of all statistical mismatches between the experimental data and model data. This cost function was used as the evolutionary algorithm's fitness function.

4.4.8 Truncation

During the algorithm run time there are up to eight new population members per generation. In order for the population to improve in fitness the less-fit individuals need to be removed. The population model used for this algorithm is a steady state model. In a steady state model the entire population is not changed at once, only part of it is [7]. For each generation new offspring and mutants are built, evaluated for fitness, and added to the existing population. The entire population is then ranked based on fitness and truncated to the initial population size. This truncation removes the less-fit individuals from the population and prevents the population from growing.

4.4.9 Termination Criteria

Every search algorithm must have termination criteria to determine when to stop searching. This algorithm has termination criteria based on the number of generations that have elapsed or on the fitness of of the best individual. After every generation the population is checked to see if the fitness is less than or equal to the number of mismatches allowed by the user. During the simulations used by this study, the acceptable number of mismatches is zero. If this criteria is met, the simulation stops, records the results in a file, and presents the user with the most fit individual.

If an individual has not been found that meets the fitness criteria, the algorithm relies on the number of generations that have elapsed. During this study, the algorithm was allowed 100 generations to find an optimal solution. The number of generations can also be increased by the operator if an answer is consistently not found. If a solution is not found the algorithm will stop and start over based upon the number of re-runs allowed.

4.4.10 Re-Runs

There are some instances where an evolutionary algorithm converges too quickly to a local minimum. Mutation operators are designed to prevent this from happening, however sometimes this cannot be avoided. The principle behind the re-runs is that if the algorithm converges too quickly it will not find a solution within the number of generations allowed. The program is restarted and the initial population is seeded with different individuals. With new individuals in the population the algorithm has a chance to take a different path to find the optimum solution.

4.5 Algorithm Flow Diagram

A flow diagram of the evolutionary algorithm is shown in Fig. 4.6. This diagram describes the individual steps performed by the algorithm to arrive at the optimal solution. The diagram begins with defining the search space and initializing the population. The next steps show the methods used for recombination and mutation. The fitness of the population is assessed, and the population is then sorted and truncated. After the new population is created, it is compared with the termination criteria. If one criterion is met, the algorithm ends. If neither criterion is not met, the algorithm continues.

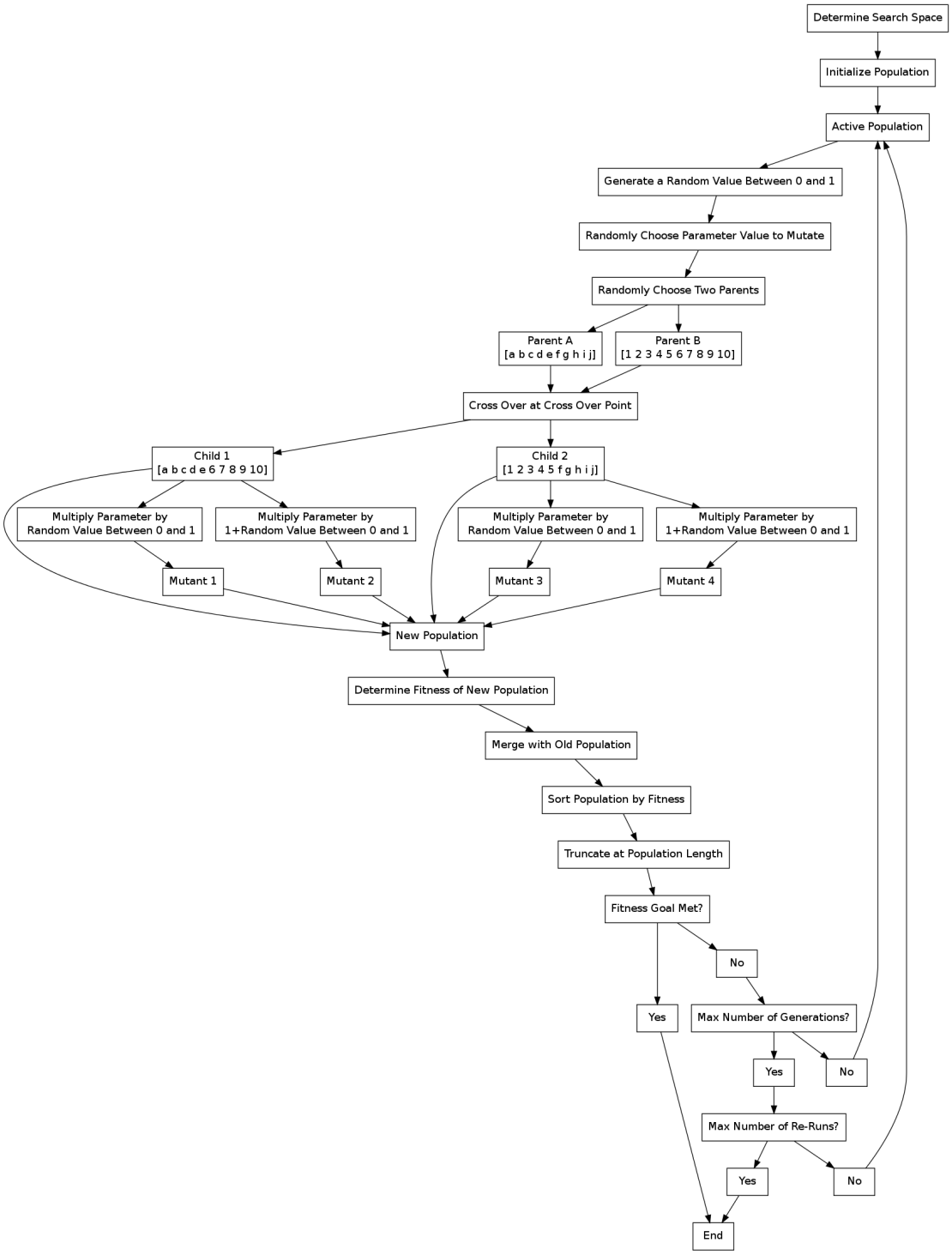

Figure 4.6: Algorithm Flow Diagram

4.6 Launching the Improved Simulation

The simulation has been reformatted in order to run in either a terminal or graphical environment. The program will automatically adjust to each environment accordingly.

4.6.1 MATLAB Desktop

In order to run the simulation in the MATLAB desktop environment the user must navigate to the root directory of the simulation. The MATLAB working directory must be changed to this root directory in order for the simulation to construct the output directories properly. If this step is not performed the log, result, and plot folders will be created in the current MATLAB directory instead.

Once the MATLAB current directory has been set to the simulation root directory, the Stability_Simulation.m file is run. The file can be run by either right-clicking on the file in the folder and selecting "Run" or by typing "Stability_Simulation" in the command window and pressing the "Enter" or "Return" key.

The simulation greets the user with a simple user interface consisting of selectable radio buttons and editable text boxes shown in Fig. 4.7. This interface is adapted from the interface designed by Andrew Sovol [1, 2] shown in Fig. 4.1. This similarity should provide an easy transition for users familiar with the previous release of this simulation software.

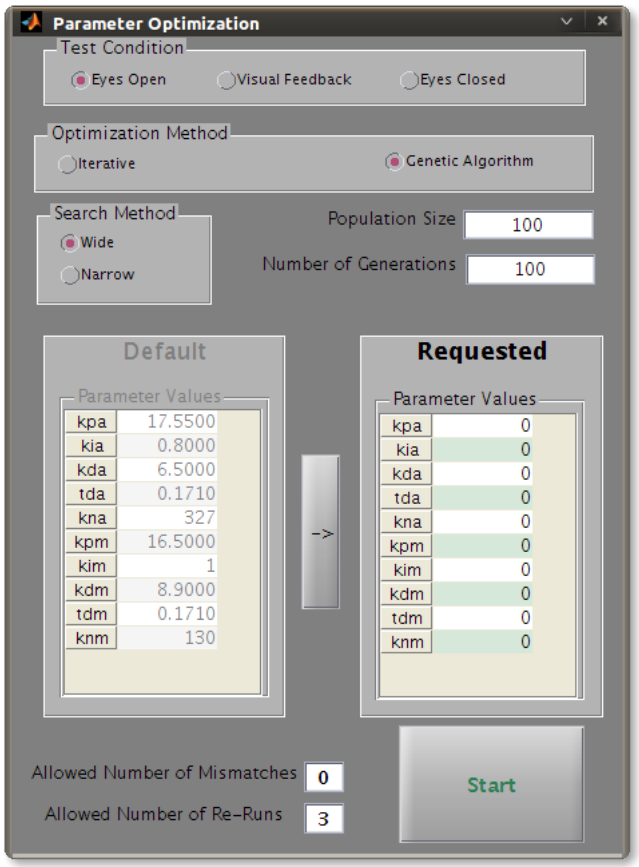

Figure 4.7: MATLAB Desktop GUI

There have been some noticeable changes to the user interface. Users are now asked for the testing condition of the patient data, “Eyes Open”, ”Eyes Closed”, or “Visual Feedback.” This improvement eliminates the need for the user to manually change the “parameters.m” and “cop1.m” files.

In order to distinguish between the two optimization methods, users are now able to select whether they wish to use the previous “Iterative Method”[1] or the new “Genetic Algorithm” method.

The user interface now includes three new text boxes, “Population Size”, “Number of Generations”, and “Allowed Number of Re-Runs.” The user can customize each of these parameters for the evolutionary algorithm to run.

As with the previous graphical user interface, users are asked to enter starting parameter values. In the event that the genetic method is selected,

and no parameters are entered, the algorithm will use default parameter values which are based on "Normal" parameter values.

Once all of the parameter values have been set in the GUI, the user presses the "Start" button. The program begins by creating directories for simulation logs, final results, and plots. In an effort to avoid any problems with the rapid accelerator in Simulink, the "slprj" (directory which contains the rapid accelerator information) is deleted upon startup. With the directory removed, the model is required to build new files on every run. The simulation begins to run, and users are greeted by a status bar which shows them the relative progress of their simulation for a single run. This status bar is shown in Fig. 4.8.

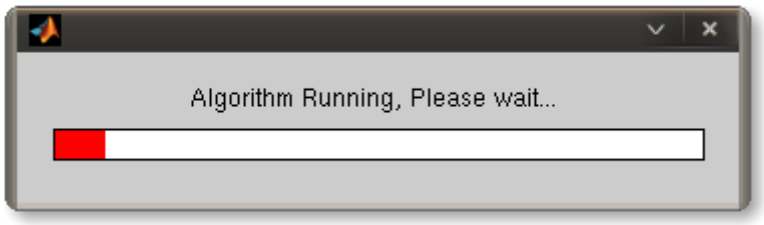

Figure 4.8: Simulation Status Bar

Every simulation generates three files for the user. In the Simulation_Logs directory, a diary entry can be found. This diary entry is the entire console output from the MATLAB command window. The second file is found under the Simulation_Results directory. This file shows the final population of the simulation, including the run time, as well as the number of generations it took to complete. The third file is a .png image of the MATLAB plot, which displays the best fitness of the population for each generation. This image is useful in determining how the algorithm progressed during the simulation. An example of a plot which completed in a single run can be found in Fig. 4.9. Each of the filenames is specific to a given run, using the date and time information in the filename.

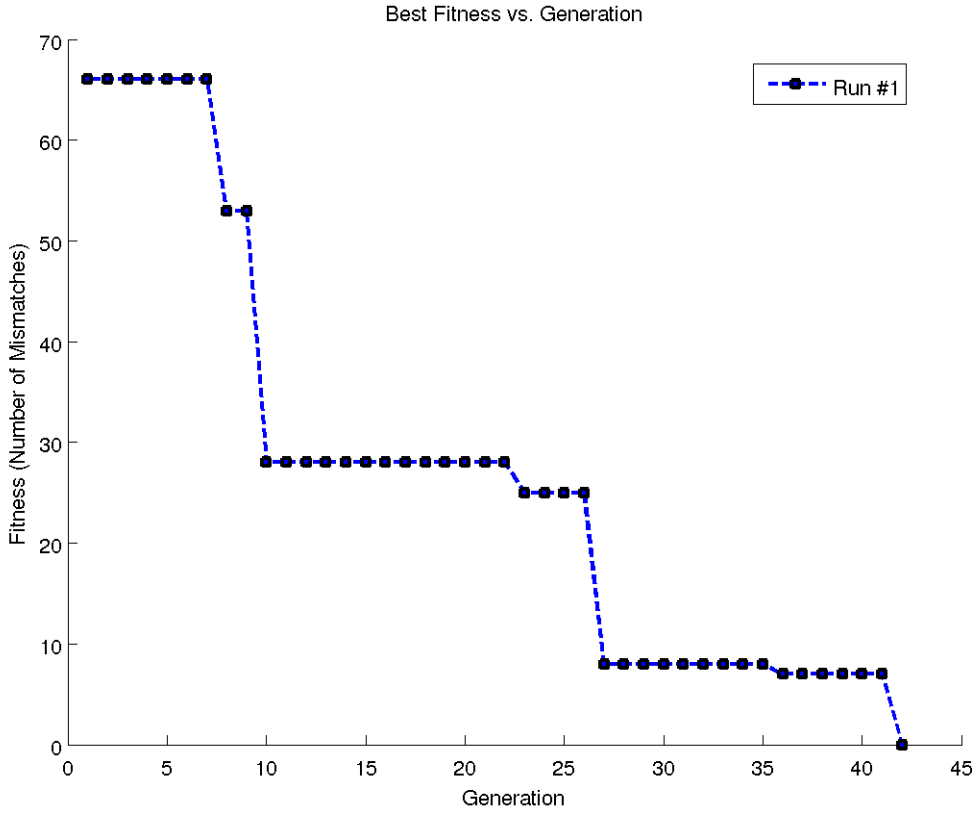

Figure 4.9: Evolutionary Algorithm Progression Plot

The iterative method designed by Andrew Sovol [1, 2] has been implemented in this version of the bi-planar stability simulation. The user must select the iterative radio button, and must also define starting conditions. No error handling has been built into this method, although the iteration time has been dramatically decreased. Solutions for these simulation runs can be found in the Simulation_Logs folder.

4.6.2 MATLAB Command Line

In some instances the MATLAB desktop environment may not be available or desired by a user. The MATLAB desktop has a large memory overhead for running editor windows and plot functions. One way to improve the performance of the model simulation is to run it on a cluster of computers such as Marquette's

Pére cluster, where command line functionality is necessary. The postural stability model simulation has been adapted to function in a command line environment if it is required.

To start MATLAB in a command line environment use the command found in Fig. 4.10.

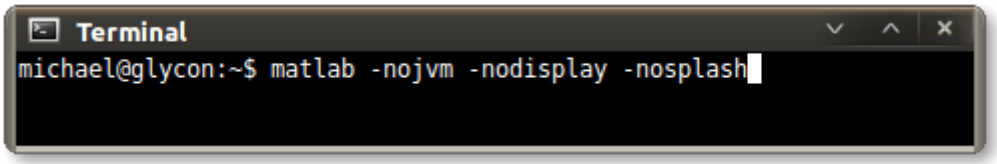

Figure 4.10: Launch Command Line MATLAB

After the above command has been entered, MATLAB will load a command line prompt and will load without any JAVA libraries. The user is greeted by the MATLAB startup screen shown in Fig. 4.11.

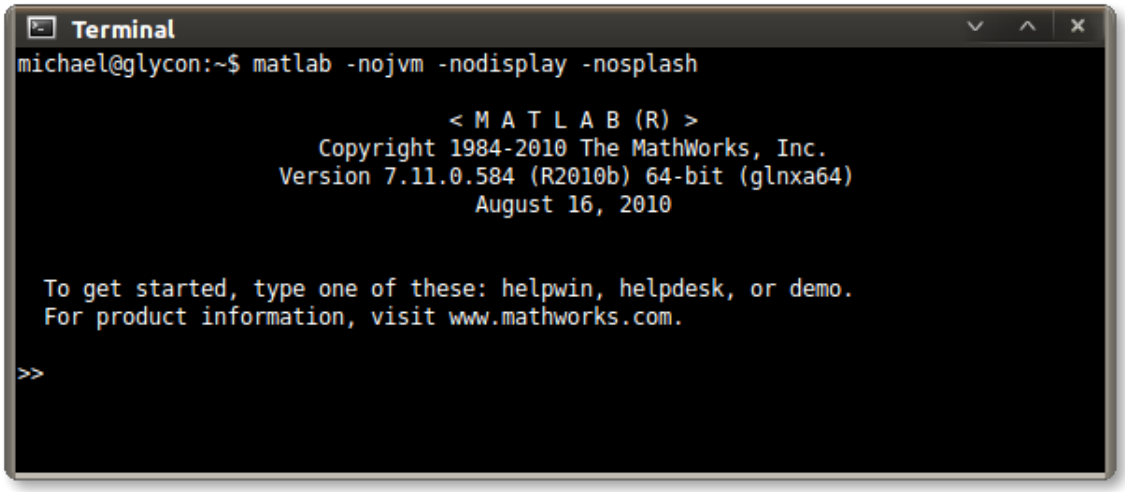

Figure 4.11: Command Line MATLAB Startup

The user must change to the root directory of the simulation folder. The change directory commands are the same as any DOS or Linux terminal, `~$ cd  path_to_directory`. After the user has changed the MATLAB current directory to the simulation root directory the simulation can be started by

entering “Stability_Simulation” into the command window, and pressing the “Enter” or “Return” key. The user will be asked to enter the parameter values for the simulation at the command prompt as shown in Fig. 4.12. Figure 4.12 shows a simulation that has begun with no user defined starting parameter values.

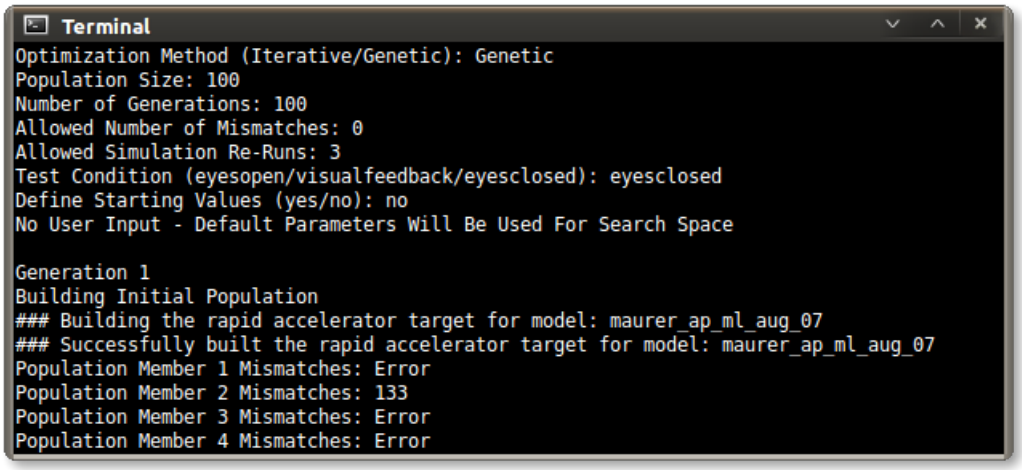

Figure 4.12: Command Line Startup With No User Defined Variables

In the situation where starting parameter values are known, the user can enter them to constrain the search space. Figure 4.13 shows a command window where the user is prompted for these parameter values.

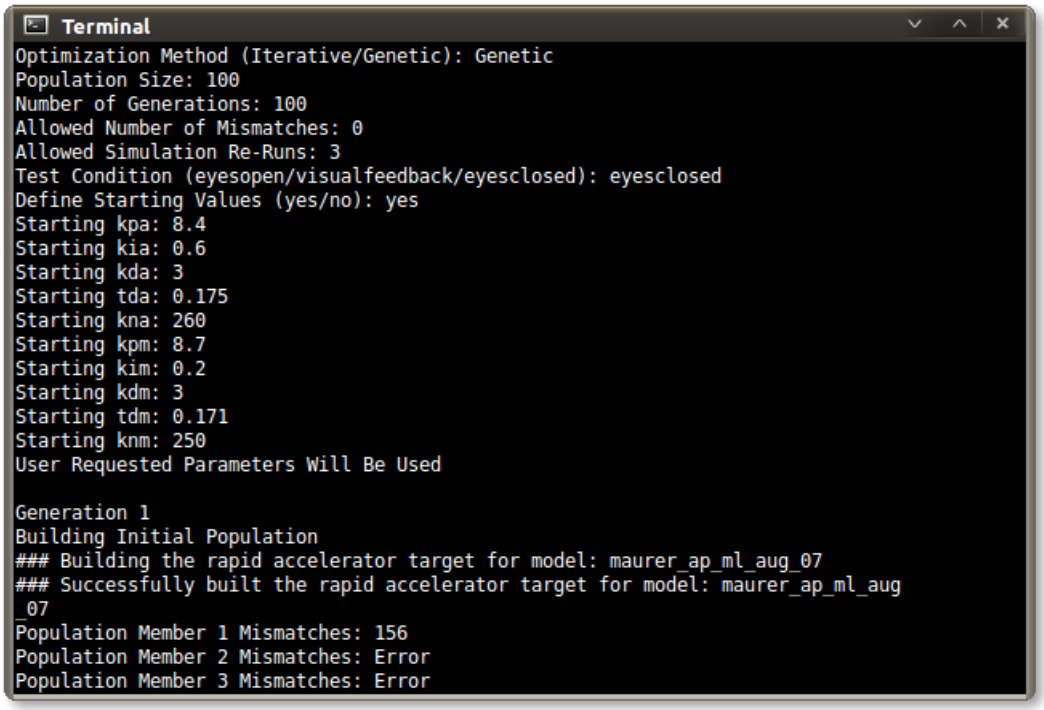

Figure 4.13: Command Line Startup With User-Defined Parameter Values

The command line version of the evolutionary algorithm also keeps a record of each simulation run. This version keeps a log file in the Simulation_Logs directory as well as a result file in the Simulation_Results directory.

There is also a command line version of the Iterative method implemented by Andrew Sovol [1, 2]. The user is greeted with a startup window, as seen in Fig. 4.14, and will choose the iterative optimization method. This method requires the user to enter initial parameter values, as no defaults have been implemented. No error handling has been added to improve this method. There has been a substantial performance improvement for this method because the iteration run time has been decreased. The iterative method does not have a result output other than the console output, however the solutions can be found in the simulation log file.

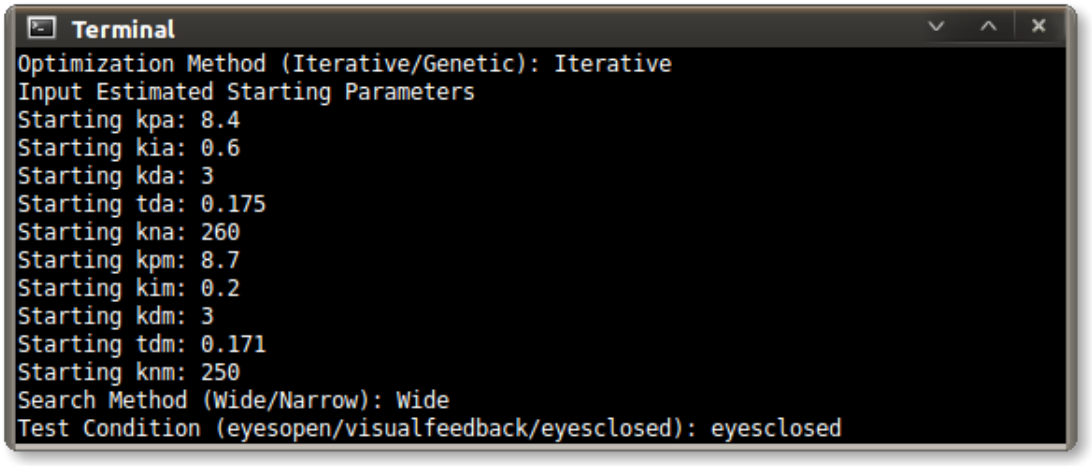

Figure 4.14: Command Line Iterative Simulation

Chapter 5

RESULTS

5.1 Testing Conditions

The bi-planar postural stability model was run using data from 17 children. The children have spastic diplegic cerebral palsy and were analyzed using the dual force plate system. For each simulation the algorithm used a population size of 100 individuals and was allowed to run for up to 100 generations. Each simulation had a maximum of 4 independent runs to converge to a solution.

5.2 Results

Simulation time was drastically improved after streamlining the existing code provided by Andrew Sovol [1, 2]. Prior to speeding up this code, a single iteration took 2 minutes and 45 seconds on average to run a single iteration. After making improvements, a single iteration ran 13.21 times faster on average. The average of a single iteration after the initial building of the rapid accelerator files is 12.49 seconds. The initial run takes 22.34 seconds because it has to build the rapid accelerator files. This longer run occurs only once per simulation trial.

The results of a series of 25 independent runs showed that 23 of the runs eventually converged to find a solution providing 0 statistical mismatches. During the series of runs there were two instances where the algorithm did

not converge to a solution within 4 tries. The average parameter values from these runs can be found in Table 5.1. It should be noted that the average time for a simulation is 6005.74 seconds, which is approximately 1 hour and 40 minutes. Figures 5.1 through 5.6 show individual parameter values during each of the 23 runs. Note that the final solution varies in each run. This data shows that there is not a unique set of parameter values for which teh model simulation produces zero statistical mismatches, but a range of values.

Average Parameter Values

Parameter	Minimum	Maximum	Mean	Standard Deviation
Kpa	8.0936	12.6075	10.16531	1.1342
Kia	0.2058	1.383	0.6847	0.45832
Kda	1.7302	3.3121	2.4178	0.45832
Tda	0.0511	0.1725	0.1071	0.03615
Kna	103.8352	543.0845	323.29319	149.2187
Kpm	6.9339	13.1569	10.04606	1.60823
Kim	0.3349	1.329	0.651067	0.343163
Kdm	2.2723	4.2186	3.068572	0.34316
Tdm	0.0857	0.2348	0.135956	0.035487
Knm	85.1511	225.0663	187.63937	30.9568
Time	1852.32	21438.6	6005.74	5768.60

Table 5.1: Average Parameter Values Over 23 Simulations

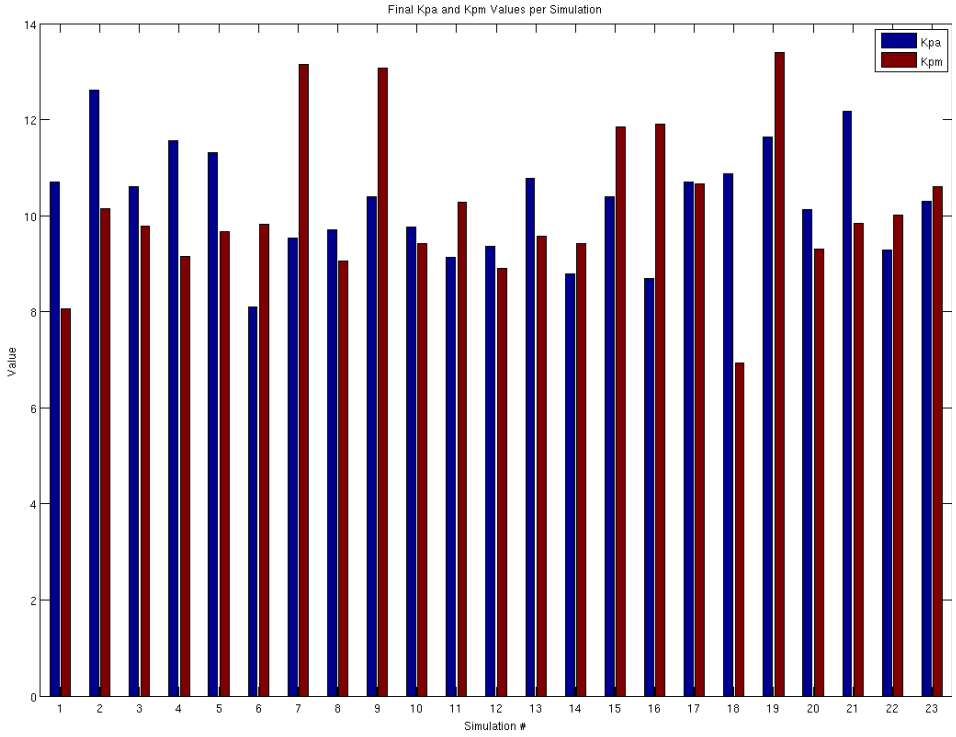

Figure 5.1: Kpa and Kpm Values ($\frac{Nm}{deg}$) Over 23 Trials

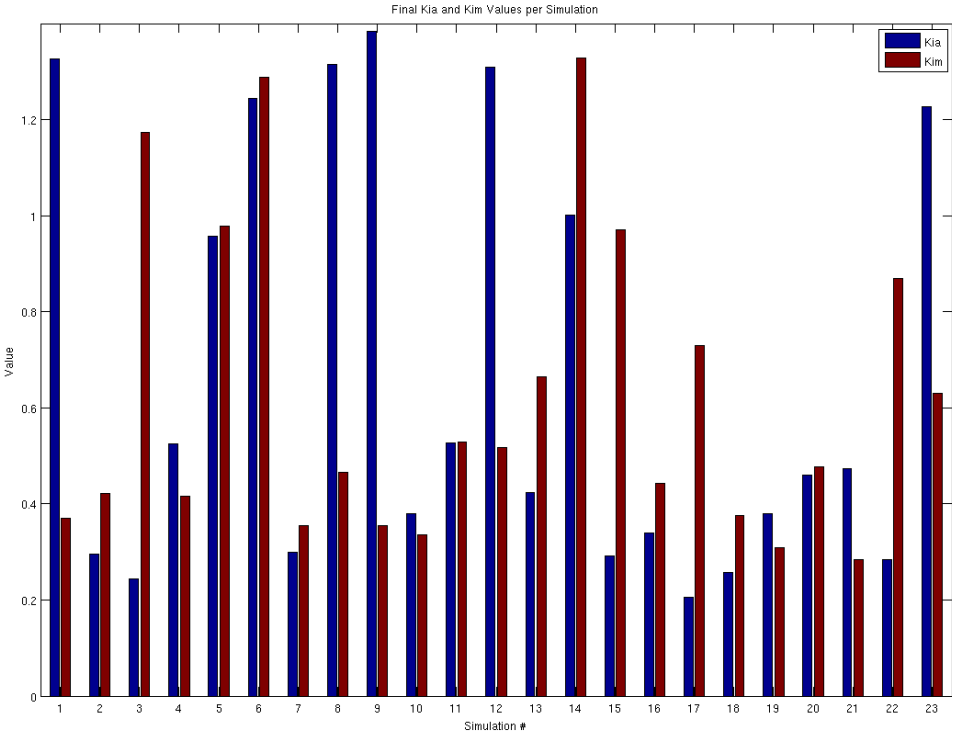

Figure 5.2: Kia and Kim Values ($\frac{Nm}{s \cdot deg}$) Over 23 Trials

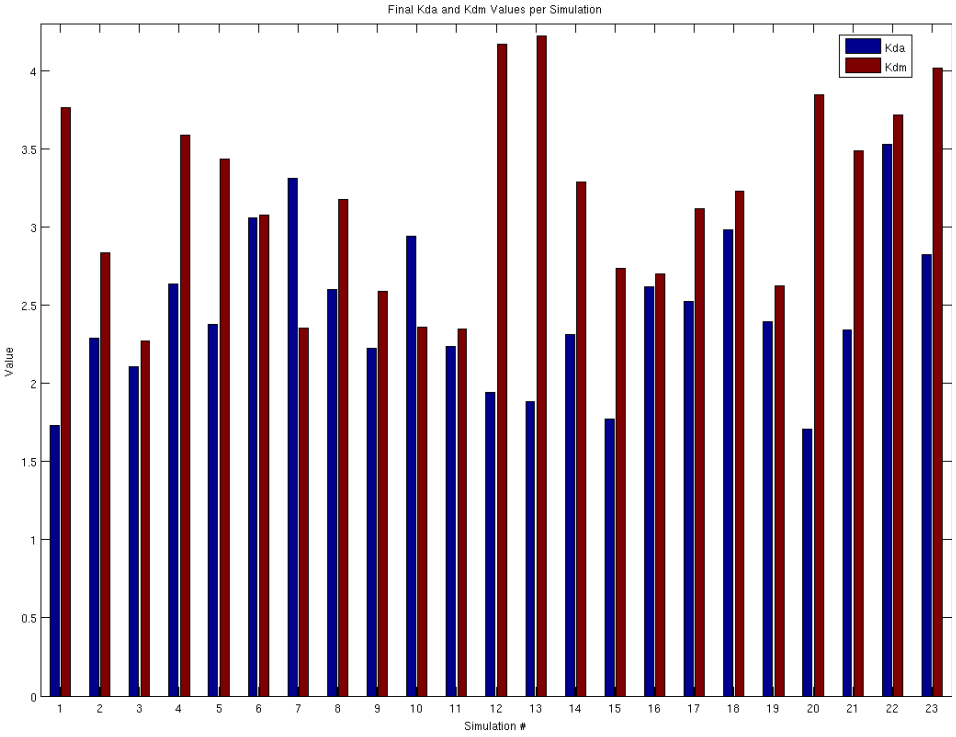

Figure 5.3: Kda and Kdm Values ($\frac{Nm \cdot s}{deg}$) Over 23 Trials

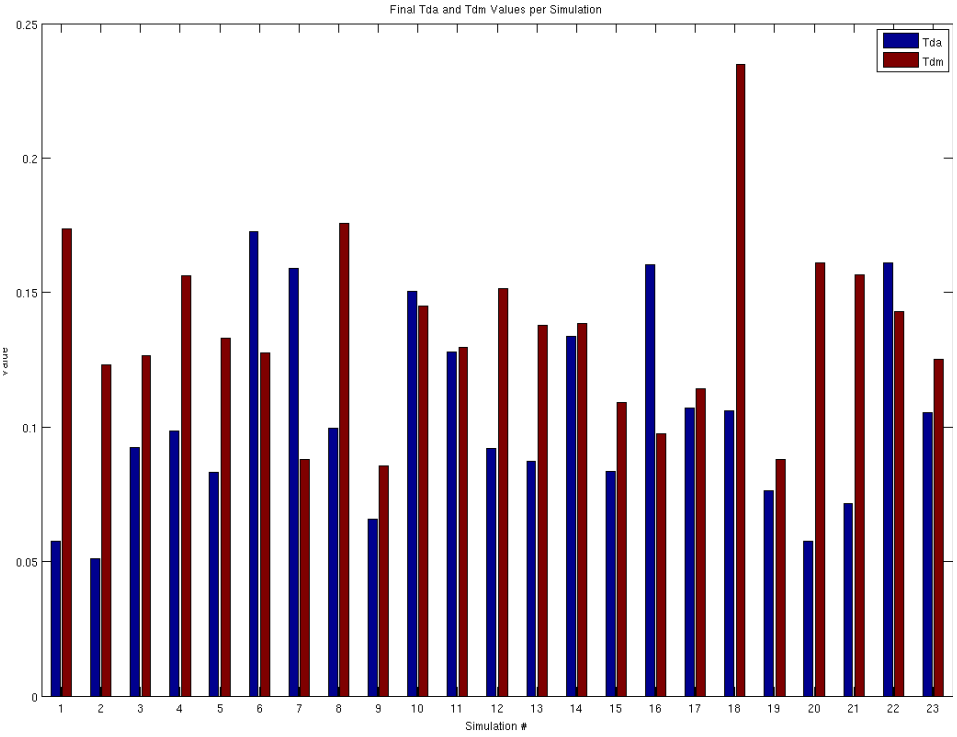

Figure 5.4: Tda and Tdm Values (s) Over 23 Trials

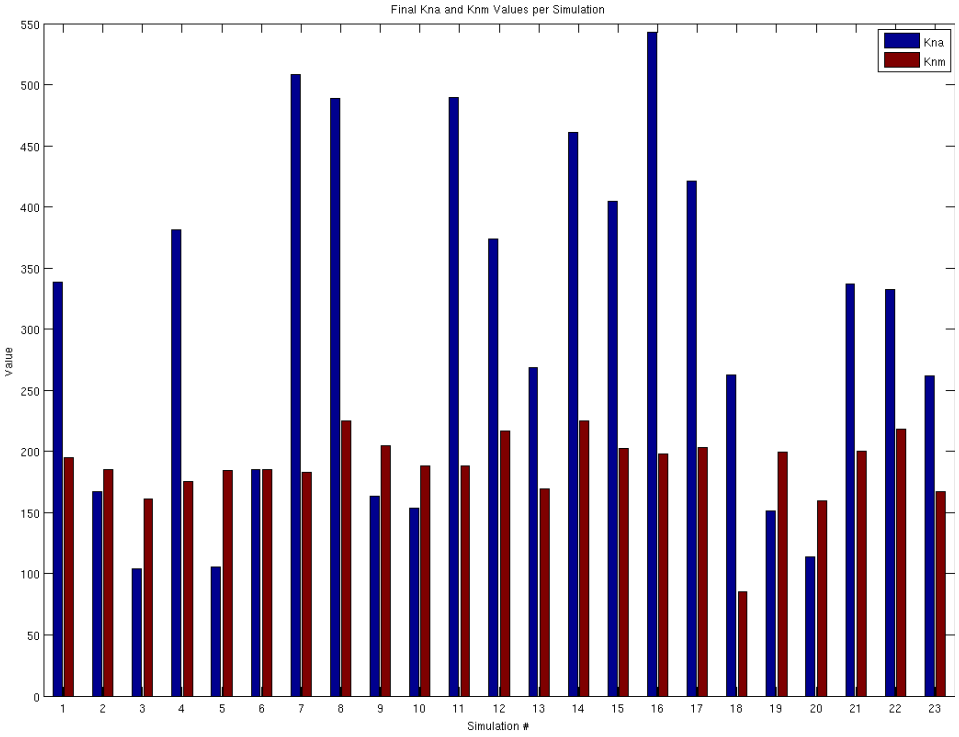

Figure 5.5: Kna and Knm Values (Gain) Over 23 Trials

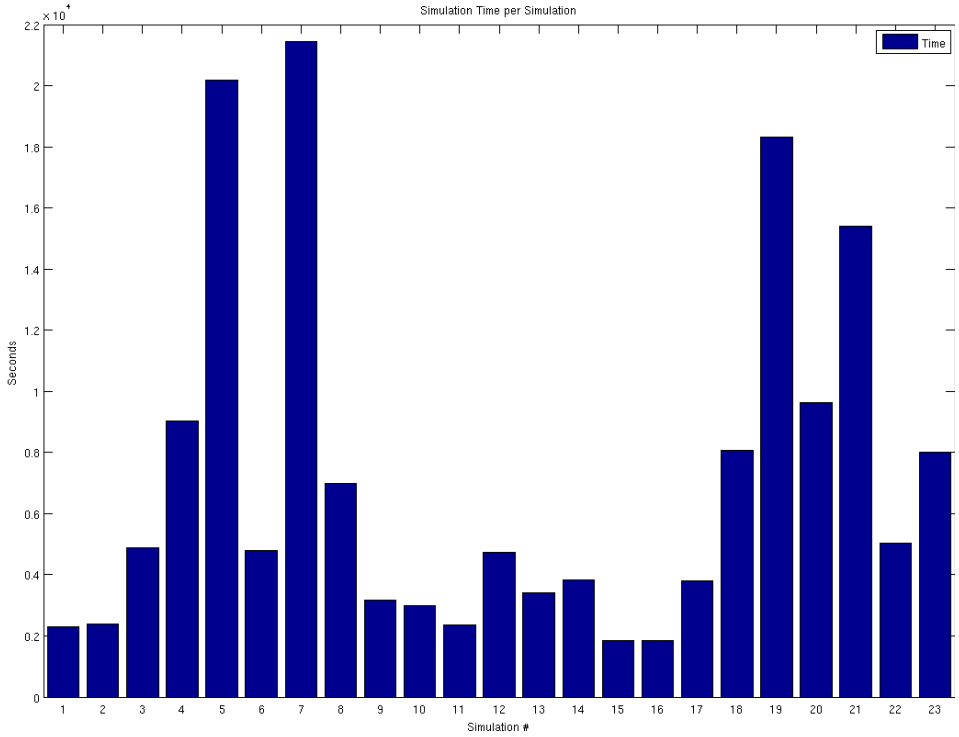

Figure 5.6: Simulation Time (s) Over 23 Trials

Chapter 6

CONCLUSION

6.1 Summary of Findings

The results of a series of 25 independent simulation runs showed that 23 of the runs eventually converged to find a solution providing 0 statistical mismatches. The average time for the algorithm to find optimal parameter values is 1 hour and 40 minutes. The fastest was slightly less than 31 minutes. Uncertainty in the parameter matching was reduced from 10% to 0.05% using this new algorithm. Table 6.1 shows the computation time required by each method to find an optimal parameter fit. All of these tests were performed on a computer using the configuration shown in Table 6.2.

	Manual	Iterative	Genetic
Computation Time	4 days	12 hours	2 hours
Typical Pace	2 weeks	1 day	$\frac{1}{2}$ day

Table 6.1: Approximate Simulation Time Comparison

6.2 Future Work

The postural stability model could benefit from future student work. One of the most important additions to the project would be a method to import new patient data. An interface in which Excel sheets can be imported directly

Processor	2.3 GHz AMD Phenom 9650 Quad-Core Processor
Motherboard	Micro-Star International 785GTM-E45
Architecture	x86-64
RAM	4 Gigabytes - DDR2 800 MHz (PC-6400)
Operating System	Ubuntu Linux 10.10
MATLAB Release	7.11.0 (R2010b)

Table 6.2: Computer Specifications

into the model would benefit the operator greatly. This method would include formatting the input data to the model's matrix requirements. Another way to improve this data input would be a seamless integration of the direct force-plate outputs to the stability model inputs.

Additional work using parallel processing could reap great benefits in reducing computation time. Marquette's Pére cluster is currently able to run a MATLAB module. A previous release of the command line simulation interface has successfully run on the head-node of the computer cluster. However, there are problems in distributing the program to other nodes in the cluster due to a MATLAB licensing problem. If Marquette purchases a license manager, this code could potentially be distributed to the cluster and a decrease in computation time could be achieved.

6.3 Conclusions

It has been proven that the initial hypothesis is true. The implementation of an evolutionary algorithm has provided this postural stability model with a faster, more accurate, and more complete method of fitting model parameter values to patient data.

Bibliography

[1] A. M. Sovol, "Bi-planar postural stability model: Fitting parameters to patient data automatically," Master's thesis, Marquette University, Milwaukee, WI, 2010.

[2] A. Sovol et al., "Bi-planar postural stability model: Fitting model parameters to patient data automatically," in *33nd Annual International Conference of the IEEE EMBS*, Buenos Aires, Argentina, 2010, pp. 3962–3965.

[3] K. Bustamante Valles et al., "Combined sagittal and coronal plane postural stability model," in *Proceedings of the 28th IEEE EMBS Annual International Conference*, New York City, USA, 2006, pp. 4576–4589.

[4] K. Bustamante Valles et al., "Application of a bi-planar postural stability model in children with cerebral palsy," in *30th Annual International Conference of the IEEE EMBS*, Vancouver, British Columbia, Canada, 2008, pp. 4535–4538.

[5] K. Bustamante Valles et al., "Using a bi-planar postural stability model to assess children with scoliosis," in *31st Annual International Conference of the IEEE EMBS*, Minneapolis, Minnesota, USA, 2009, pp. 7010–7013.

[6] K. Bustamante Valles et al., "Analysis of postural stability following posterior spinal fusion in adolescents with idiopathic scoliosis," *Studies in Heath Technology and Informatics*, pp. 127–131, 2010.

[7] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, G. Rozenberg, Ed. Berlin, Germany: Springer, 2003.

[8] K. Bustamante Valles et al., "A bi-planar model of postural sway using force plate data," October 2010, unpublished.

[9] D. Winter, "Human balance and posture controle during standing and walking," *Gait & Posture*, vol. 3, pp. 193–214, December 1995.

[10] C. Morris. (2002) Orthotic management of children with cerebral palsy. [Online]. Available: http://www.oandp.org/jpo/library/2002_04_150.asp

[11] C. Morris. (2011) Information on cerebral palsy. provided by the u.s. centers for disease control & prevention. [Online]. Available: http://www.cdc.gov/ncbddd/dd/ddcp.htm

[12] C. Good, "The genetic basis of adolescent idiopathic scoliosis," *Journal of the Spinal Research Foundation*, vol. 4, pp. 13–17, Spring 2009.

[13] NIAMS/NIH. (2008, July) Questions and answers about scoliosis in children and adolescents. [Online]. Available: http://www.niams.nih.gov/Health_Info/Scoliosis/default.asp

[14] C. Maurer et al., "Multisensory control of human upright stance," *Experimental Brain Research*, vol. 171, pp. 231–250, 2006. [Online]. Available: http://dx.doi.org/10.1007/s00221-005-0256-y

[15] P. H. Raven et al., *Biology*, 7th ed. Boston, Massachusetts: McGraw-Hill, 2005.

[16] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*. Berlin, Germany: Springer, 2008.

[17] M. Affenzeller et al., *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*, A. Sydow, Ed. Boca Raton, Florida: CRC Press, 2009.

[18] K. A. De Jong, *Evolutionary Computation: A Unified Approach*. Cambridge, Massachusetts: The MIT Press, 2006.

[19] W. Banzhaf et al., *Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. San Francisco, California: Morgan Kaufmann Publishers, Inc., 1998.

[20] C. R. Reeves and J. E. Rowe, *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory*, R. Sharda and S. Voβ, Eds. Boston, Massachusetts: Kluwer Academic Publishers, 2003.

Appendix A

Error Code Lookup

This section covers the solution for errors which may be experienced by a user during installation or running of this simulation.

A.1 Random Number Generator

Error Message:

```
??? Undefined function or method 'RandStream' for input arguments of type 'char'.

Error in ==> Stability_Simulation at 5
RandStream.setDefaultStream(RandStream('mt19937ar','seed',sum(100*clock)));
```

Solution:

This error is encountered in version of MATLAB prior to R2008b. To resolve this problem, please upgrade MATLAB to a more recent version. This can also be resolved by removing line 5 from the Stability_Simulation.m file. Removal of this line will remove the random seed generation based on the clock, and can result in duplicate runs.

A.2 Indexing

Error Message:

```
??? Attempted to access f(0); index must be a positive integer
or logical.
```

Solution:

Initial parameters entered into the OpeningGUI window are too far from being suitable for the current population. Try other sets of parameters [1].

A.3 Continuous State Derivative

Error Message:

```
Error occurred while simulating the model 'maurer_ap_ml_aug_07'
in rapid accelerator mode: Continuous state derivative of
'element 0' is non-finite.
```

Solution:

Initial parameters entered into the OpeningGUI window are too far from being suitable for the current population. Try other sets of parameters [1].

Appendix B

Source Code

B.1 Stability_Simulation.m

This file contains the majority of the necessary code for the simulation.

```
clc
clear all
close all

RandStream.setDefaultStream(RandStream('mt19937ar','seed',sum(100*clock))); % Change the stream for RAND function


slprjdirectory=exist('slprj'); %Checks for existance of rapid accelerator folder
if slprjdirectory ~= 0
rmdir('slprj','s') % This removes the compiled rapid accelerator folder after simulation.
end

logdirectory=exist('Simulation_Logs','dir');
if logdirectory~=7
    mkdir('Simulation_Logs')
end


c=fix(clock); %Set up for Log filename
date=[c(2),c(3),c(1),c(4),c(5)];
filename=sprintf('Simulation_Logs/Simulation_Log_%g_%g_%g_%g_%g.txt',date(1),date(2),date(3),date(4),date(5));
diary(filename) %Start Log File


javaon=usejava('desktop'); %Checks if java desktop is on.

if javaon==1
uiwait(OpeningGUI) %Opens GUI and waits for completion

        if populationsize<10            %Error Checking
        disp('Population Must Be Greater Than Or Equal To 10')
        diary off
        return
        end

        if numberofgenerations<=0            %Error Checking
        disp('Population Must Be Greater Than 0')
        diary off
        return
        end

else
    OptimizationMethodinput=input('Optimization Method (Iterative/Genetic): ','s');
    switch OptimizationMethodinput
        case {'iterative','Iterative','i','I'}

            disp('Input Estimated Starting Parameters')
        kpa = input('Starting kpa: ');
        kia = input('Starting kia: ');
        kda = input('Starting kda: ');
        tda = input('Starting tda: ');
        kna = input('Starting kna: ');
        kpm = input('Starting kpm: ');
        kim = input('Starting kim: ');
        kdm = input('Starting kdm: ');
        tdm = input('Starting tdm: ');
        knm = input('Starting knm: ');
        searchmethoduserinput= input('Search Method (Wide/Narrow): ','s');

        switch searchmethoduserinput
            case {'wide','Wide','W','w'}
        SearchMethod='radiobutton1';
            case {'narrow','Narrow','n','N'}
        SearchMethod='radiobutton2';
            otherwise
                disp('Iterative Search Method Not Valid - End Simulation')
```

```
            diary off
              return
        end

       testcondition=input('Test Condition (eyesopen/visualfeedback/eyesclosed): ','s');      %Not yet implemented as of 11/09/10
       InputConstraint=input('Allowed Number of Mismatches: ');
      OptimizationMethod='iterative';


        case {'genetic','Genetic','g','G'}
    populationsize=input('Population Size: ');
    if populationsize<10            %Error Checking
        disp('Population Must Be Greater Than Or Equal To 10')
        diary off
        return
    end
    numberofgenerations=input('Number of Generations: ');
        if numberofgenerations<=0            %Error Checking
        disp('Population Must Be Greater Than 0')
        diary off
        return
        end

    InputConstraint=input('Allowed Number of Mismatches: ');
    reruns=input('Allowed Simulation Re-Runs: ');

    testcondition=input('Test Condition (eyesopen/visualfeedback/eyesclosed): ','s'); %This will have to be modified eventually to more cases, only using eyesopen for now.
    userinputcondition=input('Define Starting Values (yes/no): ','s');

    switch userinputcondition
        case {'yes','y','Yes','Y'}

RequestedParameters(1) = input('Starting kpa: ');   %Define User Search Parameters - This allows a tigher search space.
RequestedParameters(2) = input('Starting kia: ');
RequestedParameters(3) = input('Starting kda: ');
RequestedParameters(4) = input('Starting tda: ');
RequestedParameters(5) = input('Starting kna: ');
RequestedParameters(6) = input('Starting kpm: ');
RequestedParameters(7) = input('Starting kim: ');
RequestedParameters(8) = input('Starting kdm: ');
RequestedParameters(9) = input('Starting tdm: ');
RequestedParameters(10) = input('Starting knm: ');

        otherwise
RequestedParameters=[0,0,0,0,0,0,0,0,0,0]; %Using this value allows future code to interpret this as default values.
    end
    OptimizationMethod='genetic';
       otherwise
       disp('Optimization Method Not Valid - End Simulation');
       diary off
       return
    end

end

switch testcondition % Make testing conditions for model
    case 'eyesopen'
        eo=1;
        eof=0;
        ec=0;
    case 'eyesclosed'
        eo=0;
        eof=0;
        ec=1;
```

```
    case 'visualfeedback'
        eo=0;
        eof=1;
        ec=0;
end

%%%%%%%%% Begin Iterative Method %%%%%%%%%%%%%%%%%%%%

switch OptimizationMethod %Determines which optimization method to run.
    case 'iterative'
        disp('Beginning Iterative Method.')

        if javaon==1
kpa = RequestedParameters(1);   %  Reads the 'RequestedParameters' matrix that is initially
defined via the front'OpeningGUI.m' panel.
kia = RequestedParameters(2);   %  This matrix is this subdivided into ten values and assigned
to the matching parameters (kpa, kia, kda, etc).
kda = RequestedParameters(3);
tda = RequestedParameters(4);
kna = RequestedParameters(5);
kpm = RequestedParameters(6);
kim = RequestedParameters(7);
kdm = RequestedParameters(8);
tdm = RequestedParameters(9);
knm = RequestedParameters(10);
        end

save('DynamicVariables.mat', 'kpa', 'kia', 'kda', 'tda', 'kna', 'kpm', 'kim', 'kdm', 'tdm',
'knm');  % Saves parameter values to 'DynamicVariables.mat' - a file that is actively written
to now, as well as after each calculation iteration.
load('StaticVariables.mat', 'eo', 'eof', 'ec', 'cpdip', 'cphem', 'J', 'm', 'h', 'g', 'noiseap',
'noiseml', 'means', 'sd', 'means_array', 'meanseof', 'sdeof', 'means_arrayeof', 'meansec',
'sdec', 'means_arrayec', 'ranges', 'ranges2', 'ranges3');  % Reads required simulation elements
from 'StaticVariables.mat' - an always-constant file that is never written to.  This read
occurs at the start of each iteration as well.

FirstIteration=1;  % Temporarily notes that the first iteration is passing through this
program.  When the program attempts to 'shift' the parameter history, this 'FirstIteration=1'
assignment will bypass that shift.

% This block of definitions initializes various variables that will be used later in the
program.
OverallLoopIteration=0;             % 'OverallLoopIteration' is initialized to zero here - it
will be incremented at the start of each calculation iteration loop to keep track of how many
calculation iterations have occurred thus far.
Bypass_TotalMismatchesOverall=0;    % 'Bypass_TotalMismatchesOverall' is initialized to zero
here.  When at zero, it enables the continuation of the calculation iteration loop.  It will
remain at zero until the 'TotalMismatchesOverall' from the previous calculation iteration are
less than or equal to that of the 'InputConstraint' defined by the user via the 'OpeningGUI.m'
window.
SpiralPosition=0;                   % Initializes the variable 'SpiralPosition', which is used
to set the initial starting position of the main logic flow.  This will increment or decrement
to different values as the main automated logic executes.
BigResults=[];                      % 'BigResults.mat' will be the final resting matrix for all
values.  Types of values recorded will include 'total mismatches' and 'current parameter
values'.  For legacy users, viewing this file after simulation is the new ouputs compared to
typing 'res' .
VariablePosition='kpa';             % Assigns the name of the first parameter automated to be
changed by the automated logic process.  This will rotate through all of the paramter names,
given an adequate amount of simulation time and free memory space.

Matrix_kpa = [0;0;0;0;0;0;0;0;0];  % Primarily GUI Use: Initialization of a history-tracking
array 'Matrix_kpa' that will retain historical values of kpa throughout each simulation
iteraion.
Matrix_kpa([1]) = [kpa];           % Primarily GUI Use: Adds the first live 'kpa' value to the
#1 matrix slot for storage.  This value will be shifted 'down' the array each time an
additional execution takes place - making space in the array for additional 'live' kpa values.
kpaCount=0;                        % Primarily GUI Use: States that the parameter 'kpa' has not
```

```
been changed yet.  If the variable 'kpa' is changed 17 times throughout a trial of simulations,
this variable should then equal 17.

Matrix_kia = [0;0;0;0;0;0;0;0;0];
Matrix_kia([1]) = [kia];
kiaCount=0;

Matrix_kda = [0;0;0;0;0;0;0;0;0];
Matrix_kda([1]) = [kda];
kdaCount=0;

Matrix_tda = [0;0;0;0;0;0;0;0;0];
Matrix_tda([1]) = [tda];
tdaCount=0;

Matrix_kna = [0;0;0;0;0;0;0;0;0];
Matrix_kna([1]) = [kna];
knaCount=0;

Matrix_kpm = [0;0;0;0;0;0;0;0;0];
Matrix_kpm([1]) = [kpm];
kpmCount=0;

Matrix_kim = [0;0;0;0;0;0;0;0;0];
Matrix_kim([1]) = [kim];
kimCount=0;

Matrix_kdm = [0;0;0;0;0;0;0;0;0];
Matrix_kdm([1]) = [kdm];
kdmCount=0;

Matrix_tdm = [0;0;0;0;0;0;0;0;0];
Matrix_tdm([1]) = [tdm];
tdmCount=0;

Matrix_knm = [0;0;0;0;0;0;0;0;0];
Matrix_knm([1]) = [knm];
knmCount=0;

PreviousParameterArray1 = [0;0;0;0;0;0;0;0;0;0];  % Primarily GUI Use: Initialization of the
'PreviousParameterArrayN' variables.  This will later store the SET of parameters that were
used N number of calculation iterations ago.
PreviousParameterArray2 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray3 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray4 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray5 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray6 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray7 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray8 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray9 = [0;0;0;0;0;0;0;0;0;0];
PreviousParameterArray10 = [0;0;0;0;0;0;0;0;0;0];

tic % Starts the stopwatch.  Will later be stopped by 'toc'.  Resulting value will be used to
display the time required to perform a single iteration of the simulation.

%=========================================================================%
%============\/ ORIGINAL CALC \/============%
%==========%

if eof==1
    means=meanseof;
    sd=sdeof;
    means_array=means_arrayeof;
end

if ec==1
    means=meansec;
    sd=sdec;
```

```
    means_array=means_arrayec;
end


 n1=1;   n2=15;
 m1=1;   m2=2;
 %noiseml=noiseap; just to check
  for i=1:7

    sim('maurer_ap_ml_aug_07',451)

    ap=cop1(tetarad_ap(:,2)); ml=cop1(tetarad_ml(:,2));  %Convert from teta to cop
    results=zeros(38,15);

    results1=calculos_simulation_june_07(ap(1:3000),ml(1:3000),'First combination');
    results(:,1)=results1;
    r1=(abs(results1-means)>sd);


    results2=calculos_simulation_june_07(ap(3001:6000),ml(3001:6000),'2 combination');
    r2=(abs(results2-means)>sd);
    results(:,2)=results2;

    results3=calculos_simulation_june_07(ap(6001:9000),ml(6001:9000),'3 combination');
    r3=(abs(results3-means)>sd);
    results(:,3)=results3;

    results4=calculos_simulation_june_07(ap(9001:12000),ml(9001:12000),'4 combination');
    r4=(abs(results4-means)>sd);
    results(:,4)=results4;

    results5=calculos_simulation_june_07(ap(12001:15000),ml(12001:15000),'5 combination');
    r5=(abs(results5-means)>sd);
    results(:,5)=results5;

    results6=calculos_simulation_june_07(ap(15001:18000),ml(15001:18000),'6 combination');
    r6=(abs(results6-means)>sd);
    results(:,6)=results6;

    results7=calculos_simulation_june_07(ap(18001:21000),ml(18001:21000),'7 combination');
    r7=(abs(results7-means)>sd);
    results(:,7)=results7;

    results8=calculos_simulation_june_07(ap(21001:24000),ml(21001:24000),'8 combination');
    r8=(abs(results8-means)>sd);
    results(:,8)=results8;

    results9=calculos_simulation_june_07(ap(24001:27000),ml(24001:27000),'9 combination');
    r9=(abs(results9-means)>sd);
    results(:,9)=results9;

    results10=calculos_simulation_june_07(ap(27001:30000),ml(27001:30000),'10 combination');
    r10=(abs(results10-means)>sd);
    results(:,10)=results10;

    results11=calculos_simulation_june_07(ap(30001:33000),ml(30001:33000),'11 combination');
    r11=(abs(results11-means)>sd);
    results(:,11)=results11;

    results12=calculos_simulation_june_07(ap(33001:36000),ml(33001:36000),'12 combination');
    r12=(abs(results12-means)>sd);
    results(:,12)=results12;

    results13=calculos_simulation_june_07(ap(36001:39000),ml(36001:39000),'13 combination');
    r13=(abs(results13-means)>sd);
    results(:,13)=results13;

    results14=calculos_simulation_june_07(ap(39001:42000),ml(39001:42000),'14 combination');
```

```
    r14=(abs(results14-means)>sd);
    results(:,14)=results14;

    results15=calculos_simulation_june_07(ap(42001:45000),ml(42001:45000),'15 combination');
    r15=(abs(results14-means)>sd);
    results(:,15)=results15;

    meanresults=mean(results,2);
    rmean=(abs(meanresults-means)>sd);
    close all;

    simulation(:,n1:n2)=results;
    n1=n1+15;
    n2=n2+15;


    res(:,m1:m2)=[meanresults,rmean];
    res1(:,i)=rmean;
   % close all
    m1=m1+2;
    m2=m2+2;

    noiseap=noiseap-1;  %initial noise seed in ap at 23340
    %noiseml=noiseap;
    noiseml=noiseml+1;  %initial noise seed in ml at 23343
end



n1=1;
n2=3;
for i=1:35
    simulationsmeans(:,i)=mean(simulation(:,n1:n2),2);
    n1=n1+3;
    n2=n2+3;
end

statsarray=simulationsmeans(:,1:17); %EDIT HERE to proper array size.  (:,1:n) ,where 'n' is↙
the array size to edit!  This value is probably set at '17' or '31' now.
overallmean=mean(statsarray,2);
statsarray=statsarray';

%display('end')

n1=1;
n2=2;
n3=3;
for i=1:35
    array(:,n1)=means_array(:,i);
    array(:,n2)=statsarray(:,i);
    array(:,n3)=zeros(17,1); %EDIT HERE to proper array size.  (n,1) ,where 'n' is the array↙
size to edit!  This value is probably set at '17' or '31' now.
    n1=n1+3;
    n2=n2+3;
    n3=n3+3;
end

highervalue=max(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.  Also,↙
'gives me the maximum value in a given metric'
lowervalue=min(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.

arr=[highervalue lowervalue];
comparison1=ranges3(:,1)-arr(:,1);
comparison2=arr(:,2)-ranges3(:,2);
[h,p]=ttest(means_array,statsarray,0.1);
arr1=[overallmean comparison1 comparison2 h'];
```

```
%==========%
%============/\ ORIGINAL CALC /\============%
%=================================================================================%

TotalMismatchesOverall = sum(sum(res(1:35,2:2:14)));

PreviousMismatchesArray = [0;0;0;0;0;0;0;0;0;0];  %Initializing a 10 iteration history array
for total mismatch tracking through iterations.
PreviousMismatchesArray([1]) = [TotalMismatchesOverall];  %Used to track overall mismatches
through a 10-deep iteration history.  Displayed via 'axes1' stem-plot in TimingGUI.m

LastChangedParameters = [0;0;0;0;0;0;0;0;0;0;];
ChangedParametersCount = [0;0;0;0;0;0;0;0;0;0;];

tElapsed=toc;  % Stops the stopwatch originally started by 'tic'.  Resulting value is assigned
to 'tElapsed' and is formatted below.  Represents the time required to perform a single
iteration of the simulation.
mm = floor(tElapsed/60);  % Formats the found tic-toc time into minutes.
ss = floor(tElapsed - floor(tElapsed/60)*60);  % Formats the found tic-toc time into seconds
SingleIterationTime=sprintf('%02d:%02d (mm:ss)' ,mm,ss);  % Displays the found tic-toc time
through a combination of the above mintues and seconds formatting.
CurrentParameters=[kpa; kia; kda; tda; kna; kpm; kim; kdm; tdm; knm];
PreviousParameterArray1 = CurrentParameters;

if javaon==1
TimingGUI %Shows TimingGUI if java desktop is on.
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Res Calc Block   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%%%%%
%

TotalMismatchesMD_RS = sum(res(1,2:2:14));
TotalMismatchesMD_AP = sum(res(2,2:2:14));
TotalMismatchesMD_ML = sum(res(3,2:2:14));
TotalMismatchesRMS_RS = sum(res(4,2:2:14));
TotalMismatchesRMS_AP = sum(res(5,2:2:14));
TotalMismatchesRMS_ML = sum(res(6,2:2:14));
TotalMismatchesRange_RS = sum(res(7,2:2:14));
TotalMismatchesRange_AP = sum(res(8,2:2:14));
TotalMismatchesRange_ML = sum(res(9,2:2:14));
TotalMismatchesTX_RS = sum(res(10,2:2:14));
TotalMismatchesTX_AP = sum(res(11,2:2:14));
TotalMismatchesTX_ML = sum(res(12,2:2:14));
TotalMismatchesMV_RS = sum(res(13,2:2:14));
TotalMismatchesMV_AP = sum(res(14,2:2:14));
TotalMismatchesMV_ML = sum(res(15,2:2:14));
TotalMismatchesMF_RS = sum(res(16,2:2:14));
TotalMismatchesMF_AP = sum(res(17,2:2:14));
TotalMismatchesMF_ML = sum(res(18,2:2:14));
TotalMismatchesSA = sum(res(19,2:2:14));
TotalMismatchesRD = sum(res(20,2:2:14));
TotalMismatchesTP_RS = sum(res(21,2:2:14));
TotalMismatchesTP_AP = sum(res(22,2:2:14));
TotalMismatchesTP_ML = sum(res(23,2:2:14));
TotalMismatchesCF_RS = sum(res(24,2:2:14));
TotalMismatchesCF_AP = sum(res(25,2:2:14));
TotalMismatchesCF_ML = sum(res(26,2:2:14));
TotalMismatchesFD_RS = sum(res(27,2:2:14));
TotalMismatchesFD_AP = sum(res(28,2:2:14));
TotalMismatchesFD_ML = sum(res(29,2:2:14));
TotalMismatchesFP50_RS = sum(res(30,2:2:14));
TotalMismatchesFP50_AP = sum(res(31,2:2:14));
TotalMismatchesFP50_ML = sum(res(32,2:2:14));
TotalMismatchesFP95_RS = sum(res(33,2:2:14));
```

```
TotalMismatchesFP95_AP = sum(res(34,2:2:14));
TotalMismatchesFP95_ML = sum(res(35,2:2:14));


%%%%%
%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Res Calc Block   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if TotalMismatchesOverall > InputConstraint;  % Verifies that the 'TotalMismatchesOverall' from↙
the previous calculation iteration are still greater than that of the 'InputConstraint' defined↙
by the user via the 'OpeningGUI.m' window.

    if SearchMethod == 'radiobutton1'  % Checks if the user-selected radio-button 'Search↙
Method' via the 'OpeningGUI.m' window has been set to 'Wide'.  If so, the 'Wide' serach method↙
is used, and parameters and explored in a more 'loose' and unconstrained manner.  Better for↙
use when parameters have not been narrowed down and general parameter fitting is still↙
relatively unknown.
        while Bypass_TotalMismatchesOverall == 0
tic % Starts the stopwatch.  Will later be stopped by 'toc'.  Resulting value will be used to↙
display the time required to perform a single iteration of the simulation.
clear eo eof ec cpdip cphem J m h g noiseap noiseml means sd means_array meanseof sdeof↙
means_arrayeof meansec sdec means_arrayec ranges ranges2 ranges3; % Clears required simulation↙
elements from the Workspace.  After this clear occurs at the start of each iteration, the↙
original values are reloaded, in the line below, to guarantee all simulation elements are pure↙
before calculatin begins.
load('StaticVariables.mat', 'eo', 'eof', 'ec', 'cpdip', 'cphem', 'J', 'm', 'h', 'g', 'noiseap',↙
'noiseml', 'means', 'sd', 'means_array', 'meanseof', 'sdeof', 'means_arrayeof', 'meansec',↙
'sdec', 'means_arrayec', 'ranges', 'ranges2', 'ranges3');  % Reads required simulation elements↙
from 'StaticVariables.mat' - an always-constant file that is never written to.
load('DynamicVariables.mat', 'kpa', 'kia', 'kda', 'tda', 'kna', 'kpm', 'kim', 'kdm', 'tdm',↙
'knm');  % Loads most recently saved parameter values from 'DynamicVariables.mat'.  The↙
previous parameter values were last saved by the most recent calculation iteration.

%=========================================================================%
%============\/ ORIGINAL CALC \/============%
%==========%

if eof==1
    means=meanseof;
    sd=sdeof;
    means_array=means_arrayeof;
end

if ec==1
    means=meansec;
    sd=sdec;
    means_array=means_arrayec;
end


 n1=1;   n2=15;
 m1=1;   m2=2;
 %noiseml=noiseap; just to check
  for i=1:7

    sim('maurer_ap_ml_aug_07',451)

    ap=cop1(tetarad_ap(:,2)); ml=cop1(tetarad_ml(:,2));  %Convert from teta to cop
    results=zeros(38,15);

    results1=calculos_simulation_june_07(ap(1:3000),ml(1:3000),'First combination');
    results(:,1)=results1;
    r1=(abs(results1-means)>sd);
```

```
results2=calculos_simulation_june_07(ap(3001:6000),ml(3001:6000),'2 combination');
r2=(abs(results2-means)>sd);
results(:,2)=results2;

results3=calculos_simulation_june_07(ap(6001:9000),ml(6001:9000),'3 combination');
r3=(abs(results3-means)>sd);
results(:,3)=results3;

results4=calculos_simulation_june_07(ap(9001:12000),ml(9001:12000),'4 combination');
r4=(abs(results4-means)>sd);
results(:,4)=results4;

results5=calculos_simulation_june_07(ap(12001:15000),ml(12001:15000),'5 combination');
r5=(abs(results5-means)>sd);
results(:,5)=results5;

results6=calculos_simulation_june_07(ap(15001:18000),ml(15001:18000),'6 combination');
r6=(abs(results6-means)>sd);
results(:,6)=results6;

results7=calculos_simulation_june_07(ap(18001:21000),ml(18001:21000),'7 combination');
r7=(abs(results7-means)>sd);
results(:,7)=results7;

results8=calculos_simulation_june_07(ap(21001:24000),ml(21001:24000),'8 combination');
r8=(abs(results8-means)>sd);
results(:,8)=results8;

results9=calculos_simulation_june_07(ap(24001:27000),ml(24001:27000),'9 combination');
r9=(abs(results9-means)>sd);
results(:,9)=results9;

results10=calculos_simulation_june_07(ap(27001:30000),ml(27001:30000),'10 combination');
r10=(abs(results10-means)>sd);
results(:,10)=results10;

results11=calculos_simulation_june_07(ap(30001:33000),ml(30001:33000),'11 combination');
r11=(abs(results11-means)>sd);
results(:,11)=results11;

results12=calculos_simulation_june_07(ap(33001:36000),ml(33001:36000),'12 combination');
r12=(abs(results12-means)>sd);
results(:,12)=results12;

results13=calculos_simulation_june_07(ap(36001:39000),ml(36001:39000),'13 combination');
r13=(abs(results13-means)>sd);
results(:,13)=results13;

results14=calculos_simulation_june_07(ap(39001:42000),ml(39001:42000),'14 combination');
r14=(abs(results14-means)>sd);
results(:,14)=results14;

results15=calculos_simulation_june_07(ap(42001:45000),ml(42001:45000),'15 combination');
r15=(abs(results14-means)>sd);
results(:,15)=results15;

meanresults=mean(results,2);
rmean=(abs(meanresults-means)>sd);
close all;

simulation(:,n1:n2)=results;
n1=n1+15;
n2=n2+15;


res(:,m1:m2)=[meanresults,rmean];
res1(:,i)=rmean;
% close all
```

```
    m1=m1+2;
    m2=m2+2;

    noiseap=noiseap-1;  %initial noise seed in ap at 23340
    %noiseml=noiseap;
    noiseml=noiseml+1;  %initial noise seed in ml at 23343
end



n1=1;
n2=3;
for i=1:35
    simulationsmeans(:,i)=mean(simulation(:,n1:n2),2);
    n1=n1+3;
    n2=n2+3;
end

statsarray=simulationsmeans(:,1:17); %EDIT HERE to proper array size.  (:,1:n) ,where 'n' is↙
the array size to edit!  This value is probably set at '17' or '31' now.
overallmean=mean(statsarray,2);
statsarray=statsarray';

%display('end')

n1=1;
n2=2;
n3=3;
for i=1:35
    array(:,n1)=means_array(:,i);
    array(:,n2)=statsarray(:,i);
    array(:,n3)=zeros(17,1); %EDIT HERE to proper array size.  (n,1) ,where 'n' is the array↙
size to edit!  This value is probably set at '17' or '31' now.
    n1=n1+3;
    n2=n2+3;
    n3=n3+3;
end

highervalue=max(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.  Also,↙
'gives me the maximum value in a given metric'
lowervalue=min(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.

arr=[highervalue lowervalue];
comparison1=ranges3(:,1)-arr(:,1);
comparison2=arr(:,2)-ranges3(:,2);
[h,p]=ttest(means_array,statsarray,0.1);
arr1=[overallmean comparison1 comparison2 h'];
%==========%
%===========/\ ORIGINAL CALC /\===========%
%=========================================================================%

TotalMismatchesOverall = sum(sum(res(1:35,2:2:14)));

if FirstIteration==0  % Checks to see this is the first iteration ('FirstIteration==1), wherein↙
the following 'shift' of parameter history code will NOT execute.
    PreviousMismatchesArray([2 3 4 5 6 7 8 9 10]) = [PreviousMismatchesArray(1)↙
PreviousMismatchesArray(2) PreviousMismatchesArray(3) PreviousMismatchesArray(4)↙
PreviousMismatchesArray(5) PreviousMismatchesArray(6) PreviousMismatchesArray(7)↙
PreviousMismatchesArray(8) PreviousMismatchesArray(9)];  % Shifts the last nine of the↙
'PreviousMismatchesArray#' values down to make room for the current value, as seen in the line↙
below.
    PreviousMismatchesArray([1]) = [TotalMismatchesOverall];  % Shifts the curent total number↙
of calculated mismatches into the ten-value array.

    PreviousParameterArray10 = PreviousParameterArray9;  % All 'PreviousParameterArray#'↙
assignments are only of value for 'TimingGUI.m' visual display.
```

```
    PreviousParameterArray9 = PreviousParameterArray8;   % Shifts all ten parameters in
'PreivousParameterArray9' into 'PreviousParameterArray10'.
    PreviousParameterArray8 = PreviousParameterArray7;   % Since only the last ten values of
each parameter are kept (PreviousParameterArray#), the values in 'PreviousParameterArray10' are
lost upon the next shift.
    PreviousParameterArray7 = PreviousParameterArray6;   % Regardless of any parameter history
kept or not kept in 'PreviousParameterArray#', ALL current and previous parameter data is
stored in 'BigResults.m' until the program is stopped and re-executed (wherein the parameter
data is over-written).
    PreviousParameterArray6 = PreviousParameterArray5;
    PreviousParameterArray5 = PreviousParameterArray4;
    PreviousParameterArray4 = PreviousParameterArray3;
    PreviousParameterArray3 = PreviousParameterArray2;
    PreviousParameterArray2 = PreviousParameterArray1;
    PreviousParameterArray1 = [kpa; kia; kda; tda; kna; kpm; kim; kdm; tdm; knm];
end

FirstIteration=0;  % Enables the ten parameter history 'shift' above.  This line is only
curtial after the very first iteration, when 'FirstIteration==1'.

OverallLoopIteration=OverallLoopIteration+1;
tElapsed=toc;  % Stops the stopwatch originally started by 'tic'.  Resulting value is assigned
to 'tElapsed' and is formatted below.  Represents the time required to perform a single
iteration of the simulation.
mm = floor(tElapsed/60);  % Formats the found tic-toc time into minutes.
ss = floor(tElapsed - floor(tElapsed/60)*60);  % Formats the found tic-toc time into seconds
SingleIterationTime=sprintf('%02d:%02d (mm:ss)' ,mm,ss);  % Displays the found tic-toc time
through a combination of the above mintues and seconds formatting.
CurrentParameters=[kpa; kia; kda; tda; kna; kpm; kim; kdm; tdm; knm];

ComboMatrix=[sum(sum(res(1:35,2:2:14))) kpa kia kda tda kna kpm kim kdm tdm knm];

BigResults=cat(1,BigResults,ComboMatrix);

save('BigResults.mat', 'BigResults');

if javaon==1
TimingGUI %Shows TimingGUI if java desktop is on.
end

switch TotalMismatchesOverall
            case 0
                display('ALL FINISHED')
                Bypass_TotalMismatchesOverall=1;
            otherwise
                display('Mis-matches remaining')
                display('Value for TotalMismatchesOverall @ >3')

                        switch VariablePosition
                             case 'kpa'
                                switch SpiralPosition
                                   case {0,1,2,3,4,  11,12,13,14}
                                       display('KPA SpiralPosition is 0 --> 4 or 11 --> 14')
                                       kpa=kpa+0.15;  % The 0.15 increment here represents 1%
of an average KPA value.  This can be adjusted for further programming control.
                                       kpaCount=kpaCount+1;
                                       SpiralPosition=SpiralPosition+1;

                                   case {5}
                                       display('KPA SpiralPosition is 5')
                                       kpa=kpa-0.9;
                                       kpaCount=kpaCount+1;
                                       SpiralPosition=SpiralPosition+1;

                                   case {6,7,8,9,  16,17,18,19}
                                       display('KPA SpiralPosition is 6 --> 9 or 16 --> 19')
                                       kpa=kpa-0.15;  % The 0.15 decrement here represents 1%
of an average KPA value.  This can be adjusted for further programming control.
```

```
                                    kpaCount=kpaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('KPA SpiralPosition is 10')
                                    kpa=kpa+1.65;
                                    kpaCount=kpaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('KPA SpiralPosition is 15')
                                    kpa=kpa-2.4;
                                    kpaCount=kpaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('KPA SpiralPosition is > 19 - trying different
parameter')
                                    kpa=kpa+1.5;
                                    SpiralPosition=0;
                                    VariablePosition='kia';
                            end

                        case 'kia'
                            switch SpiralPosition
                                case {0,1,2,3,4,  11,12,13,14}
                                    display('KIA SpiralPosition is 0 --> 4 or 11 --> 14')
                                    kia=kia+0.01;  % The 0.01 increment here represents 1%
of an average KIA value.  This can be adjusted for further programming control.
                                    kiaCount=kiaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {5}
                                    display('KIA SpiralPosition is 5')
                                    kia=kia-0.06;
                                    kiaCount=kiaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {6,7,8,9,  16,17,18,19}
                                    display('KIA SpiralPosition is 6 --> 9 or 16 --> 19')
                                    kia=kia-0.01;  % The 0.01 decrement here represents 1%
of an average KIA value.  This can be adjusted for further programming control.
                                    kiaCount=kiaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('KIA SpiralPosition is 10')
                                    kia=kia+0.11;
                                    kiaCount=kiaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('KIA SpiralPosition is 15')
                                    kia=kia-0.16;
                                    kiaCount=kiaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('KIA SpiralPosition is > 19 - trying different
parameter')
                                    kia=kia+0.1;
                                    SpiralPosition=0;
                                    VariablePosition='kda';
                            end

                        case 'kda'
                            switch SpiralPosition
                                case {0,1,2,3,4,  11,12,13,14}
```

```
                                    display('KDA SpiralPosition is 0 --> 4 or 11 --> 14')
                                    kda=kda+0.05;  % The 0.05 increment here represents 1%↙
of an average KDA value.  This can be adjusted for further programming control.
                                    kdaCount=kdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {5}
                                    display('KDA SpiralPosition is 5')
                                    kda=kda-0.3;
                                    kdaCount=kdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {6,7,8,9,  16,17,18,19}
                                    display('KDA SpiralPosition is 6 --> 9 or 16 --> 19')
                                    kda=kda-0.05;  % The 0.05 decrement here represents 1%↙
of an average KDA value.  This can be adjusted for further programming control.
                                    kdaCount=kdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('KDA SpiralPosition is 10')
                                    kda=kda+0.55;
                                    kdaCount=kdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('KDA SpiralPosition is 15')
                                    kda=kda-0.8;
                                    kdaCount=kdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('KDA SpiralPosition is > 19 - trying different↙
parameter')
                                    kda=kda+0.5;
                                    SpiralPosition=0;
                                    VariablePosition='tda';
                            end

                        case 'tda'
                            switch SpiralPosition
                                case {0,1,2,3,4,  11,12,13,14}
                                    display('TDA SpiralPosition is 0 --> 4 or 11 --> 14')
                                    tda=tda+0.0015;  % The 0.0015 increment here represents↙
1% of an average TDA value.  This can be adjusted for further programming control.
                                    tdaCount=tdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {5}
                                    display('TDA SpiralPosition is 5')
                                    tda=tda-0.009;
                                    tdaCount=tdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {6,7,8,9,  16,17,18,19}
                                    display('TDA SpiralPosition is 6 --> 9 or 16 --> 19')
                                    tda=tda-0.0015;  % The 0.0015 decrement here represents↙
1% of an average TDA value.  This can be adjusted for further programming control.
                                    tdaCount=tdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('TDA SpiralPosition is 10')
                                    tda=tda+0.0165;
                                    tdaCount=tdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
```

```
                                    display('TDA SpiralPosition is 15')
                                    tda=tda-0.024;
                                    tdaCount=tdaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('TDA SpiralPosition is > 19 - trying different
parameter')
                                    tda=tda+0.015;
                                    SpiralPosition=0;
                                    VariablePosition='kna';
                            end

                        case 'kna'
                            switch SpiralPosition
                                case {0,1,2,3,4,  11,12,13,14}
                                    display('KNA SpiralPosition is 0 --> 4 or 11 --> 14')
                                    kna=kna+3;  % The 3.0 increment here represents 1% of an
average KNA value.  This can be adjusted for further programming control.
                                    knaCount=knaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {5}
                                    display('KNA SpiralPosition is 5')
                                    kna=kna-18;
                                    knaCount=knaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {6,7,8,9,  16,17,18,19}
                                    display('KNA SpiralPosition is 6 --> 9 or 16 --> 19')
                                    kna=kna-3;  % The 3.0 decrement here represents 1% of an
average KNA value.  This can be adjusted for further programming control.
                                    knaCount=knaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('KNA SpiralPosition is 10')
                                    kna=kna+33;
                                    knaCount=knaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('KNA SpiralPosition is 15')
                                    kna=kna-48;
                                    knaCount=knaCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('KNA SpiralPosition is > 19 - trying different
parameter')
                                    kna=kna+30;
                                    SpiralPosition=0;
                                    VariablePosition='kpm';
                            end

                        case 'kpm'
                            switch SpiralPosition
                                case {0,1,2,3,4,  11,12,13,14}
                                    display('KPM SpiralPosition is 0 --> 4 or 11 --> 14')
                                    kpm=kpm+0.15;  % The 0.15 increment here represents 1%
of an average KPM value.  This can be adjusted for further programming control.
                                    kpmCount=kpmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {5}
                                    display('KPM SpiralPosition is 5')
                                    kpm=kpm-0.9;
                                    kpmCount=kpmCount+1;
```

```
                                    SpiralPosition=SpiralPosition+1;

                                case {6,7,8,9,  16,17,18,19}
                                    display('KPM SpiralPosition is 6 --> 9 or 16 --> 19')
                                    kpm=kpm-0.15;  % The 0.15 decrement here represents 1%↙
of an average KPM value.  This can be adjusted for further programming control.
                                    kpmCount=kpmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('KPM SpiralPosition is 10')
                                    kpm=kpm+1.65;
                                    kpmCount=kpmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('KPM SpiralPosition is 15')
                                    kpm=kpm-2.4;
                                    kpmCount=kpmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('KPM SpiralPosition is > 19 - trying different↙
parameter')
                                    kpm=kpm+1.5;
                                    SpiralPosition=0;
                                    VariablePosition='kim';
                            end

                        case 'kim'
                            switch SpiralPosition
                                case {0,1,2,3,4,  11,12,13,14}
                                    display('KIM SpiralPosition is 0 --> 4 or 11 --> 14')
                                    kim=kim+0.01;  % The 0.01 increment here represents 1%↙
of an average KIM value.  This can be adjusted for further programming control.
                                    kimCount=kimCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {5}
                                    display('KIM SpiralPosition is 5')
                                    kim=kim-0.06;
                                    kimCount=kimCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {6,7,8,9,  16,17,18,19}
                                    display('KIM SpiralPosition is 6 --> 9 or 16 --> 19')
                                    kim=kim-0.01;  % The 0.01 decrement here represents 1%↙
of an average KIM value.  This can be adjusted for further programming control.
                                    kimCount=kimCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('KIM SpiralPosition is 10')
                                    kim=kim+0.11;
                                    kimCount=kimCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('KIM SpiralPosition is 15')
                                    kim=kim-0.16;
                                    kimCount=kimCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('KIM SpiralPosition is > 19 - trying different↙
parameter')
                                    kim=kim+0.1;
                                    SpiralPosition=0;
```

```
                                VariablePosition='kdm';
                            end

                        case 'kdm'
                            switch SpiralPosition
                               case {0,1,2,3,4,  11,12,13,14}
                                    display('KDM SpiralPosition is 0 --> 4 or 11 --> 14')
                                    kdm=kdm+0.05;  % The 0.05 increment here represents 1%
of an average KDM value.  This can be adjusted for further programming control.
                                    kdmCount=kdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               case {5}
                                    display('KDM SpiralPosition is 5')
                                    kdm=kdm-0.3;
                                    kdmCount=kdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               case {6,7,8,9,  16,17,18,19}
                                    display('KDM SpiralPosition is 6 --> 9 or 16 --> 19')
                                    kdm=kdm-0.05;  % The 0.05 decrement here represents 1%
of an average KDM value.  This can be adjusted for further programming control.
                                    kdmCount=kdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               case {10}
                                    display('KDM SpiralPosition is 10')
                                    kdm=kdm+0.55;
                                    kdmCount=kdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               case {15}
                                    display('KDM SpiralPosition is 15')
                                    kdm=kdm-0.8;
                                    kdmCount=kdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               otherwise
                                    display('KDM SpiralPosition is > 19 - trying different
parameter')
                                    kdm=kdm+0.5;
                                    SpiralPosition=0;
                                    VariablePosition='tdm';
                            end

                        case 'tdm'
                            switch SpiralPosition
                               case {0,1,2,3,4,  11,12,13,14}
                                    display('TDM SpiralPosition is 0 --> 4 or 11 --> 14')
                                    tdm=tdm+0.0015;  % The 0.0015 increment here represents
1% of an average TDM value.  This can be adjusted for further programming control.
                                    tdmCount=tdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               case {5}
                                    display('TDM SpiralPosition is 5')
                                    tdm=tdm-0.009;
                                    tdmCount=tdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               case {6,7,8,9,  16,17,18,19}
                                    display('TDM SpiralPosition is 6 --> 9 or 16 --> 19')
                                    tdm=tdm-0.0015;  % The 0.0015 decrement here represents
1% of an average TDM value.  This can be adjusted for further programming control.
                                    tdmCount=tdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                               case {10}
```

```
                                    display('TDM SpiralPosition is 10')
                                    tdm=tdm+0.0165;
                                    tdmCount=tdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('TDM SpiralPosition is 15')
                                    tdm=tdm-0.024;
                                    tdmCount=tdmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('TDM SpiralPosition is > 19 - trying different↙
parameter')
                                    tdm=tdm+0.015;
                                    SpiralPosition=0;
                                    VariablePosition='knm';
                            end

                        case 'knm'
                            switch SpiralPosition
                                case {0,1,2,3,4,  11,12,13,14}
                                    display('KNM SpiralPosition is 0 --> 4 or 11 --> 14')
                                    knm=knm+1.5;  % The 1.5 increment here represents 1% of↙
an average KNM value.  This can be adjusted for further programming control.
                                    knmCount=knmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {5}
                                    display('KNM SpiralPosition is 5')
                                    knm=knm-9;
                                    knmCount=knmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {6,7,8,9,  16,17,18,19}
                                    display('KNM SpiralPosition is 6 --> 9 or 16 --> 19')
                                    knm=knm-1.5;  % The 1.5 decrement here represents 1% of↙
an average KNM value.  This can be adjusted for further programming control.
                                    knmCount=knmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {10}
                                    display('KNM SpiralPosition is 10')
                                    knm=knm+16.5;
                                    knmCount=knmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                case {15}
                                    display('KNM SpiralPosition is 15')
                                    knm=knm-24;
                                    knmCount=knmCount+1;
                                    SpiralPosition=SpiralPosition+1;

                                otherwise
                                    display('KNM SpiralPosition is > 19 - trying different↙
parameter')
                                    knm=knm+15;
                                    SpiralPosition=0;
                                    %end of variable switching....  bypassing
                                    Bypass_TotalMismatchesOverall=1;
                            end

                        otherwise
                           display('Variable case selected does not have a case!')
                    end

                   %The following records the last/previous set of parameters - primarily↙
used for UI display.
```

```
                        LastChangedParameters = [Matrix_kpa(1); Matrix_kia(1); Matrix_kda(1);↙
Matrix_tda(1); Matrix_kna(1); Matrix_kpm(1); Matrix_kim(1); Matrix_kdm(1); Matrix_tdm(1);↙
Matrix_knm(1)];

                        ChangedParametersCount = [kpaCount; kiaCount; kdaCount; tdaCount;↙
knaCount; kpmCount; kimCount; kdmCount; tdmCount; knmCount];

                        %The following increments all 'parameter history' arrays to include the↙
newest parameter that has recently been updated.
                        %This shifts all 10 parameter values down (the initial '10th parameter↙
value' is lost) and the newest parameter that has
                        %recently been updated fills the '1' spot in the array.
                        Matrix_kpa([2 3 4 5 6 7 8 9 10]) = [Matrix_kpa(1) Matrix_kpa(2)↙
Matrix_kpa(3) Matrix_kpa(4) Matrix_kpa(5) Matrix_kpa(6) Matrix_kpa(7) Matrix_kpa(8) Matrix_kpa↙
(9)];
                        Matrix_kpa([1]) = [kpa];
                        Matrix_kpm([2 3 4 5 6 7 8 9 10]) = [Matrix_kpm(1) Matrix_kpm(2)↙
Matrix_kpm(3) Matrix_kpm(4) Matrix_kpm(5) Matrix_kpm(6) Matrix_kpm(7) Matrix_kpm(8) Matrix_kpm↙
(9)];
                        Matrix_kpm([1]) = [kpm];
                        Matrix_kia([2 3 4 5 6 7 8 9 10]) = [Matrix_kia(1) Matrix_kia(2)↙
Matrix_kia(3) Matrix_kia(4) Matrix_kia(5) Matrix_kia(6) Matrix_kia(7) Matrix_kia(8) Matrix_kia↙
(9)];
                        Matrix_kia([1]) = [kia];
                        Matrix_kim([2 3 4 5 6 7 8 9 10]) = [Matrix_kim(1) Matrix_kim(2)↙
Matrix_kim(3) Matrix_kim(4) Matrix_kim(5) Matrix_kim(6) Matrix_kim(7) Matrix_kim(8) Matrix_kim↙
(9)];
                        Matrix_kim([1]) = [kim];
                        Matrix_kda([2 3 4 5 6 7 8 9 10]) = [Matrix_kda(1) Matrix_kda(2)↙
Matrix_kda(3) Matrix_kda(4) Matrix_kda(5) Matrix_kda(6) Matrix_kda(7) Matrix_kda(8) Matrix_kda↙
(9)];
                        Matrix_kda([1]) = [kda];
                        Matrix_kdm([2 3 4 5 6 7 8 9 10]) = [Matrix_kdm(1) Matrix_kdm(2)↙
Matrix_kdm(3) Matrix_kdm(4) Matrix_kdm(5) Matrix_kdm(6) Matrix_kdm(7) Matrix_kdm(8) Matrix_kdm↙
(9)];
                        Matrix_kdm([1]) = [kdm];
                        Matrix_tda([2 3 4 5 6 7 8 9 10]) = [Matrix_tda(1) Matrix_tda(2)↙
Matrix_tda(3) Matrix_tda(4) Matrix_tda(5) Matrix_tda(6) Matrix_tda(7) Matrix_tda(8) Matrix_tda↙
(9)];
                        Matrix_tda([1]) = [tda];
                        Matrix_tdm([2 3 4 5 6 7 8 9 10]) = [Matrix_tdm(1) Matrix_tdm(2)↙
Matrix_tdm(3) Matrix_tdm(4) Matrix_tdm(5) Matrix_tdm(6) Matrix_tdm(7) Matrix_tdm(8) Matrix_tdm↙
(9)];
                        Matrix_tdm([1]) = [tdm];
                        Matrix_kna([2 3 4 5 6 7 8 9 10]) = [Matrix_kna(1) Matrix_kna(2)↙
Matrix_kna(3) Matrix_kna(4) Matrix_kna(5) Matrix_kna(6) Matrix_kna(7) Matrix_kna(8) Matrix_kna↙
(9)];
                        Matrix_kna([1]) = [kna];
                        Matrix_knm([2 3 4 5 6 7 8 9 10]) = [Matrix_knm(1) Matrix_knm(2)↙
Matrix_knm(3) Matrix_knm(4) Matrix_knm(5) Matrix_knm(6) Matrix_knm(7) Matrix_knm(8) Matrix_knm↙
(9)];
                        Matrix_knm([1]) = [knm];

                        save('DynamicVariables.mat', 'kpa', 'kia', 'kda', 'tda', 'kna', 'kpm',↙
'kim', 'kdm', 'tdm', 'knm');

                disp(sprintf('kpa Loop finished for %dx time',OverallLoopIteration));
        end
        end
    else
    end

    if SearchMethod == 'radiobutton2'  % Checks if the user-selected radio-button 'Search↙
Method' via the 'OpeningGUI.m' window has been set to 'Narrow'.  If so, the 'Narrow' serach↙
method is used, and parameters and explored in a more 'tight' and constrained manner.  Better↙
for use when parameters have already been narrowed down significantly.
        while Bypass_TotalMismatchesOverall == 0
tic % Starts the stopwatch.  Will later be stopped by 'toc'.  Resulting value will be used to↙
```

```
display the time required to perform a single iteration of the simulation.
clear eo eof ec cpdip cphem J m h g noiseap noiseml means sd means_array meanseof sdeof
means_arrayeof meansec sdec means_arrayec ranges ranges2 ranges3;  % Clears required simulation
elements from the Workspace.  After this clear occurs at the start of each iteration, the
original values are reloaded, in the line below, to guarantee all simulation elements are pure
before calculatin begins.
load('StaticVariables.mat', 'eo', 'eof', 'ec', 'cpdip', 'cphem', 'J', 'm', 'h', 'g', 'noiseap',
'noiseml', 'means', 'sd', 'means_array', 'meanseof', 'sdeof', 'means_arrayeof', 'meansec',
'sdec', 'means_arrayec', 'ranges', 'ranges2', 'ranges3');  % Reads required simulation elements
from 'StaticVariables.mat' - an always-constant file that is never written to.
load('DynamicVariables.mat', 'kpa', 'kia', 'kda', 'tda', 'kna', 'kpm', 'kim', 'kdm', 'tdm',
'knm');  % Loads most recently saved parameter values from 'DynamicVariables.mat'.  The
previous parameter values were last saved by the most recent calculation iteration.

%=========================================================================%
%============\/ ORIGINAL CALC \/============%
%==========%

if eof==1
    means=meanseof;
    sd=sdeof;
    means_array=means_arrayeof;
end

if ec==1
    means=meansec;
    sd=sdec;
    means_array=means_arrayec;
end


 n1=1;    n2=15;
 m1=1;    m2=2;
 %noiseml=noiseap; just to check
  for i=1:7

    sim('maurer_ap_ml_aug_07',451)

    ap=cop1(tetarad_ap(:,2)); ml=cop1(tetarad_ml(:,2));  %Convert from teta to cop
    results=zeros(38,15);

    results1=calculos_simulation_june_07(ap(1:3000),ml(1:3000),'First combination');
    results(:,1)=results1;
    r1=(abs(results1-means)>sd);


    results2=calculos_simulation_june_07(ap(3001:6000),ml(3001:6000),'2 combination');
    r2=(abs(results2-means)>sd);
    results(:,2)=results2;

    results3=calculos_simulation_june_07(ap(6001:9000),ml(6001:9000),'3 combination');
    r3=(abs(results3-means)>sd);
    results(:,3)=results3;

    results4=calculos_simulation_june_07(ap(9001:12000),ml(9001:12000),'4 combination');
    r4=(abs(results4-means)>sd);
    results(:,4)=results4;

    results5=calculos_simulation_june_07(ap(12001:15000),ml(12001:15000),'5 combination');
    r5=(abs(results5-means)>sd);
    results(:,5)=results5;

    results6=calculos_simulation_june_07(ap(15001:18000),ml(15001:18000),'6 combination');
    r6=(abs(results6-means)>sd);
    results(:,6)=results6;

    results7=calculos_simulation_june_07(ap(18001:21000),ml(18001:21000),'7 combination');
    r7=(abs(results7-means)>sd);
```

```
    results(:,7)=results7;

    results8=calculos_simulation_june_07(ap(21001:24000),ml(21001:24000),'8 combination');
    r8=(abs(results8-means)>sd);
    results(:,8)=results8;

    results9=calculos_simulation_june_07(ap(24001:27000),ml(24001:27000),'9 combination');
    r9=(abs(results9-means)>sd);
    results(:,9)=results9;

    results10=calculos_simulation_june_07(ap(27001:30000),ml(27001:30000),'10 combination');
    r10=(abs(results10-means)>sd);
    results(:,10)=results10;

    results11=calculos_simulation_june_07(ap(30001:33000),ml(30001:33000),'11 combination');
    r11=(abs(results11-means)>sd);
    results(:,11)=results11;

    results12=calculos_simulation_june_07(ap(33001:36000),ml(33001:36000),'12 combination');
    r12=(abs(results12-means)>sd);
    results(:,12)=results12;

    results13=calculos_simulation_june_07(ap(36001:39000),ml(36001:39000),'13 combination');
    r13=(abs(results13-means)>sd);
    results(:,13)=results13;

    results14=calculos_simulation_june_07(ap(39001:42000),ml(39001:42000),'14 combination');
    r14=(abs(results14-means)>sd);
    results(:,14)=results14;

    results15=calculos_simulation_june_07(ap(42001:45000),ml(42001:45000),'15 combination');
    r15=(abs(results14-means)>sd);
    results(:,15)=results15;

    meanresults=mean(results,2);
    rmean=(abs(meanresults-means)>sd);
    close all;

    simulation(:,n1:n2)=results;
    n1=n1+15;
    n2=n2+15;


    res(:,m1:m2)=[meanresults,rmean];
    res1(:,i)=rmean;
   % close all
    m1=m1+2;
    m2=m2+2;

    noiseap=noiseap-1;  %initial noise seed in ap at 23340
    %noiseml=noiseap;
    noiseml=noiseml+1;  %initial noise seed in ml at 23343
end



n1=1;
n2=3;
for i=1:35
    simulationsmeans(:,i)=mean(simulation(:,n1:n2),2);
    n1=n1+3;
    n2=n2+3;
end

statsarray=simulationsmeans(:,1:17); %EDIT HERE to proper array size.  (:,1:n) ,where 'n' is↙
the array size to edit!  This value is probably set at '17' or '31' now.
overallmean=mean(statsarray,2);
statsarray=statsarray';
```

```
%display('end')

n1=1;
n2=2;
n3=3;
for i=1:35
    array(:,n1)=means_array(:,i);
    array(:,n2)=statsarray(:,i);
    array(:,n3)=zeros(17,1); %EDIT HERE to proper array size.  (n,1) ,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.
    n1=n1+3;
    n2=n2+3;
    n3=n3+3;
end

highervalue=max(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2) ,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.  Also, 'gives me the maximum value in a given metric'
lowervalue=min(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2) ,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.

arr=[highervalue lowervalue];
comparison1=ranges3(:,1)-arr(:,1);
comparison2=arr(:,2)-ranges3(:,2);
[h,p]=ttest(means_array,statsarray,0.1);
arr1=[overallmean comparison1 comparison2 h'];
%==========%
%===========/\ ORIGINAL CALC /\===========%
%=========================================================================%

TotalMismatchesOverall = sum(sum(res(1:35,2:2:14)));

if FirstIteration==0  % Checks to see this is the first iteration ('FirstIteration==1), wherein the following 'shift' of parameter history code will NOT execute.
    PreviousMismatchesArray([2 3 4 5 6 7 8 9 10]) = [PreviousMismatchesArray(1) PreviousMismatchesArray(2) PreviousMismatchesArray(3) PreviousMismatchesArray(4) PreviousMismatchesArray(5) PreviousMismatchesArray(6) PreviousMismatchesArray(7) PreviousMismatchesArray(8) PreviousMismatchesArray(9)];  % Shifts the last nine of the 'PreviousMismatchesArray#' values down to make room for the current value, as seen in the line below.
    PreviousMismatchesArray([1]) = [TotalMismatchesOverall];  % Shifts the curent total number of calculated mismatches into the ten-value array.

    PreviousParameterArray10 = PreviousParameterArray9;  % All 'PreviousParameterArray#' assignments are only of value for 'TimingGUI.m' visual display.
    PreviousParameterArray9 = PreviousParameterArray8;   % Shifts all ten parameters in 'PreivousParameterArray9' into 'PreviousParameterArray10'.
    PreviousParameterArray8 = PreviousParameterArray7;   % Since only the last ten values of each parameter are kept (PreviousParameterArray#), the values in 'PreviousParameterArray10' are lost upon the next shift.
    PreviousParameterArray7 = PreviousParameterArray6;   % Regardless of any parameter history kept or not kept in 'PreviousParameterArray#', ALL current and previous parameter data is stored in 'BigResults.m' until the program is stopped and re-executed (wherein the parameter data is over-written).
    PreviousParameterArray6 = PreviousParameterArray5;
    PreviousParameterArray5 = PreviousParameterArray4;
    PreviousParameterArray4 = PreviousParameterArray3;
    PreviousParameterArray3 = PreviousParameterArray2;
    PreviousParameterArray2 = PreviousParameterArray1;
    PreviousParameterArray1 = [kpa; kia; kda; tda; kna; kpm; kim; kdm; tdm; knm];
end

FirstIteration=0;  % Enables the ten parameter history 'shift' above.  This line is only curtial after the very first iteration, when 'FirstIteration==1'.

OverallLoopIteration=OverallLoopIteration+1;
tElapsed=toc;  % Stops the stopwatch originally started by 'tic'.  Resulting value is assigned
```

```
to 'tElapsed' and is formatted below.  Represents the time required to perform a single
iteration of the simulation.
mm = floor(tElapsed/60);
ss = floor(tElapsed - floor(tElapsed/60)*60);
SingleIterationTime=sprintf('%02d:%02d (mm:ss)' ,mm,ss);
CurrentParameters=[kpa; kia; kda; tda; kna; kpm; kim; kdm; tdm; knm];

ComboMatrix=[sum(sum(res(1:35,2:2:14))) kpa kia kda tda kna kpm kim kdm tdm knm];

BigResults=cat(1,BigResults,ComboMatrix);

save('BigResults.mat', 'BigResults');

if javaon==1
TimingGUI  %Displays TimingGUI if in java desktop
end

switch TotalMismatchesOverall
            case 0
                display('ALL FINSHED')
                Bypass_TotalMismatchesOverall=1;
            otherwise
                display('Mis-matches remaining')
                display('Value for TotalMismatchesOverall @ > InputConstraint variable')

switch OverallLoopIteration
    case
{1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,
67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99,101,103,105,107,109,111,113,115,117,119,121,
123,125,127,129,131,133,135,137,139,141,143,145,147,149,151,153,155,157,159,161,163,165,167,169
,171,173,175,177,179,181,183,185,187,189,191,193,195,197,199,201,203,205,207,209,211,213,215,21
7,219,221,223,225,227,229,231,233,235,237,239,241,243,245,247,249,251,253,255}
        display('lsb')
        kpa=RequestedParameters(1)+0.15;  % The 0.15 increment here represents 1% of an average
KPA value.  This can be adjusted for further programming control.
        kpaCount=kpaCount+1;

    otherwise
        display('not true for lsb')
        kpa=RequestedParameters(1);
end


switch OverallLoopIteration
    case
{2,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,66,70,74,78,82,86,90,94,98,102,106,110,114,118,1
22,126,130,134,138,142,146,150,154,158,162,166,170,174,178,182,186,190,194,198,202,206,210,214,
218,222,226,230,234,238,242,246,250,254,3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63,67,71,75,
79,83,87,91,95,99,103,107,111,115,119,123,127,131,135,139,143,147,151,155,159,163,167,171,175,1
79,183,187,191,195,199,203,207,211,215,219,223,227,231,235,239,243,247,251,255}
        display('2nd lsb')
        kia=RequestedParameters(2)+0.01;  % The 0.01 increment here represents 1% of an average
KIA value.  This can be adjusted for further programming control.
        kiaCount=kiaCount+1;

    otherwise
        display('not true for 2nd lsb')
        kia=RequestedParameters(2);
end


switch OverallLoopIteration
    case
{4,12,20,28,36,44,52,60,68,76,84,92,100,108,116,124,132,140,148,156,164,172,180,188,196,204,212
,220,228,236,244,252,5,13,21,29,37,45,53,61,69,77,85,93,101,109,117,125,133,141,149,157,165,173
,181,189,197,205,213,221,229,237,245,253,6,14,22,30,38,46,54,62,70,78,86,94,102,110,118,126,134
,142,150,158,166,174,182,190,198,206,214,222,230,238,246,254,7,15,23,31,39,47,55,63,71,79,87,95
,103,111,119,127,135,143,151,159,167,175,183,191,199,207,215,223,231,239,247,255}
```

```
        display('3rd lsb')
        kda=RequestedParameters(3)+0.05;  % The 0.05 increment here represents 1% of an average
KDA value.  This can be adjusted for further programming control.
        kdaCount=kdaCount+1;

    otherwise
        display('not true for 3rd lsb')
        kda=RequestedParameters(3);
end


switch OverallLoopIteration
    case
{8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31,40,41,42,43,44,45,46,47,56,57,58,59,60,61,62,63,
72,73,74,75,76,77,78,79,88,89,90,91,92,93,94,95,104,105,106,107,108,109,110,111,120,121,122,123
,124,125,126,127,136,137,138,139,140,141,142,143,152,153,154,155,156,157,158,159,168,169,170,17
1,172,173,174,175,184,185,186,187,188,189,190,191,200,201,202,203,204,205,206,207,216,217,218,2
19,220,221,222,223,232,233,234,235,236,237,238,239,248,249,250,251,252,253,254,255}
        display('4th lsb')
        tda=RequestedParameters(4)+0.0015;  % The 0.0015 increment here represents 1% of an
average TDA value.  This can be adjusted for further programming control.
        tdaCount=tdaCount+1;

    otherwise
        display('not ture for 4th lsb')
        tda=RequestedParameters(4);
end


switch OverallLoopIteration
    case
{16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,6
3,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,112,113,114,115,116,117,118,119,120,121,122,1
23,124,125,126,127,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,176,177,178,
179,180,181,182,183,184,185,186,187,188,189,190,191,208,209,210,211,212,213,214,215,216,217,218
,219,220,221,222,223,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255}
        display('5th lsb')
        kna=RequestedParameters(5)+3;  % The 3.0 increment here represents 1% of an average KNA
value.  This can be adjusted for further programming control.
        knaCount=knaCount+1;

    otherwise
        display('not ture for 5th lsb')
        kna=RequestedParameters(5);
end


switch OverallLoopIteration
    case
{32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,6
3,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,1
20,121,122,123,124,125,126,127,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,
176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,224,225,226,227,228,229,230,231
,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,25
5}
        display('6th lsb')
        kpm=RequestedParameters(6)+0.15;  % The 0.15 increment here represents 1% of an average
KPM value.  This can be adjusted for further programming control.
        kpmCount=kpmCount+1;

    otherwise
        display('not ture for 6th lsb')
        kpm=RequestedParameters(6);
end


switch OverallLoopIteration
    case
```

```
{64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,9↙
5,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,1↙
20,121,122,123,124,125,126,127,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,↙
208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231↙
,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,25↙
5}
        display('7th lsb')
        kim=RequestedParameters(7)+0.01;  % The 0.01 increment here represents 1% of an average↙
KIM value.  This can be adjusted for further programming control.
        kimCount=kimCount+1;

    otherwise
        display('not ture for 7th lsb')
        kim=RequestedParameters(7);
end


switch OverallLoopIteration
    case↙
{128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,15↙
1,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,1↙
75,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,↙
199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222↙
,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,24↙
6,247,248,249,250,251,252,253,254,255}
        display('8th lsb')
        kdm=RequestedParameters(8)+0.05;  % The 0.05 increment here represents 1% of an average↙
KDM value.  This can be adjusted for further programming control.
        kdmCount=kdmCount+1;

    otherwise
        display('not ture for 8th lsb')
        kdm=RequestedParameters(8);
end

                        %The following records the last/previous set of parameters - primarily↙
used for UI display.
                        LastChangedParameters = [Matrix_kpa(1); Matrix_kia(1); Matrix_kda(1);↙
Matrix_tda(1); Matrix_kna(1); Matrix_kpm(1); Matrix_kim(1); Matrix_kdm(1); Matrix_tdm(1);↙
Matrix_knm(1)];


                        ChangedParametersCount = [kpaCount; kiaCount; kdaCount; tdaCount;↙
knaCount; kpmCount; kimCount; kdmCount; tdmCount; knmCount];

                        %The following increments all 'parameter history' arrays to include the↙
newest parameter that has recently been updated.
                        %This shifts all 10 parameter values down (the initial '10th parameter↙
value' is lost) and the newest parameter that has
                        %recently been updated fills the '1' spot in the array.
                        Matrix_kpa([2 3 4 5 6 7 8 9 10]) = [Matrix_kpa(1) Matrix_kpa(2)↙
Matrix_kpa(3) Matrix_kpa(4) Matrix_kpa(5) Matrix_kpa(6) Matrix_kpa(7) Matrix_kpa(8) Matrix_kpa↙
(9)];
                        Matrix_kpa([1]) = [kpa];
                        Matrix_kpm([2 3 4 5 6 7 8 9 10]) = [Matrix_kpm(1) Matrix_kpm(2)↙
Matrix_kpm(3) Matrix_kpm(4) Matrix_kpm(5) Matrix_kpm(6) Matrix_kpm(7) Matrix_kpm(8) Matrix_kpm↙
(9)];
                        Matrix_kpm([1]) = [kpm];
                        Matrix_kia([2 3 4 5 6 7 8 9 10]) = [Matrix_kia(1) Matrix_kia(2)↙
Matrix_kia(3) Matrix_kia(4) Matrix_kia(5) Matrix_kia(6) Matrix_kia(7) Matrix_kia(8) Matrix_kia↙
(9)];
                        Matrix_kia([1]) = [kia];
                        Matrix_kim([2 3 4 5 6 7 8 9 10]) = [Matrix_kim(1) Matrix_kim(2)↙
Matrix_kim(3) Matrix_kim(4) Matrix_kim(5) Matrix_kim(6) Matrix_kim(7) Matrix_kim(8) Matrix_kim↙
(9)];
                        Matrix_kim([1]) = [kim];
                        Matrix_kda([2 3 4 5 6 7 8 9 10]) = [Matrix_kda(1) Matrix_kda(2)↙
Matrix_kda(3) Matrix_kda(4) Matrix_kda(5) Matrix_kda(6) Matrix_kda(7) Matrix_kda(8) Matrix_kda↙
```

```
(9)];
                        Matrix_kda([1]) = [kda];
                        Matrix_kdm([2 3 4 5 6 7 8 9 10]) = [Matrix_kdm(1) Matrix_kdm(2)↙
Matrix_kdm(3) Matrix_kdm(4) Matrix_kdm(5) Matrix_kdm(6) Matrix_kdm(7) Matrix_kdm(8) Matrix_kdm↙
(9)];
                        Matrix_kdm([1]) = [kdm];
                        Matrix_tda([2 3 4 5 6 7 8 9 10]) = [Matrix_tda(1) Matrix_tda(2)↙
Matrix_tda(3) Matrix_tda(4) Matrix_tda(5) Matrix_tda(6) Matrix_tda(7) Matrix_tda(8) Matrix_tda↙
(9)];
                        Matrix_tda([1]) = [tda];
                        Matrix_tdm([2 3 4 5 6 7 8 9 10]) = [Matrix_tdm(1) Matrix_tdm(2)↙
Matrix_tdm(3) Matrix_tdm(4) Matrix_tdm(5) Matrix_tdm(6) Matrix_tdm(7) Matrix_tdm(8) Matrix_tdm↙
(9)];
                        Matrix_tdm([1]) = [tdm];
                        Matrix_kna([2 3 4 5 6 7 8 9 10]) = [Matrix_kna(1) Matrix_kna(2)↙
Matrix_kna(3) Matrix_kna(4) Matrix_kna(5) Matrix_kna(6) Matrix_kna(7) Matrix_kna(8) Matrix_kna↙
(9)];
                        Matrix_kna([1]) = [kna];
                        Matrix_knm([2 3 4 5 6 7 8 9 10]) = [Matrix_knm(1) Matrix_knm(2)↙
Matrix_knm(3) Matrix_knm(4) Matrix_knm(5) Matrix_knm(6) Matrix_knm(7) Matrix_knm(8) Matrix_knm↙
(9)];
                        Matrix_knm([1]) = [knm];

                        save('DynamicVariables.mat', 'kpa', 'kia', 'kda', 'tda', 'kna', 'kpm',↙
'kim', 'kdm', 'tdm', 'knm');

                disp(sprintf('kpa Loop finished for %dx time',OverallLoopIteration));
        end
        end
    else
    end


else
end
display('Simulation has stopped.'); %Will display in the console only when all calculations↙
have completed cleanly AND the cost function has been minizied within the operator's specified↙
requirements (this minimization requirement is listed as user-defined variable↙
'InputConstraint').

if OverallLoopIteration==0
display('Final Simulation Results');
fprintf('Mismatches = %g\n',TotalMismatchesOverall)
fprintf('kpa = %g\n',kpa)
fprintf('kia = %g\n',kia)
fprintf('kda = %g\n',kda)
fprintf('tda = %g\n',tda)
fprintf('kna = %g\n',kna)
fprintf('kpm = %g\n',kpm)
fprintf('kim = %g\n',kim)
fprintf('kdm = %g\n',kdm)
fprintf('tdm = %g\n',tdm)
else
display('Final Simulation Results');
fprintf('Mismatches = %g\n',ComboMatrix(1))
fprintf('kpa = %g\n',ComboMatrix(2))
fprintf('kia = %g\n',ComboMatrix(3))
fprintf('kda = %g\n',ComboMatrix(4))
fprintf('tda = %g\n',ComboMatrix(5))
fprintf('kna = %g\n',ComboMatrix(6))
fprintf('kpm = %g\n',ComboMatrix(7))
fprintf('kim = %g\n',ComboMatrix(8))
fprintf('kdm = %g\n',ComboMatrix(9))
fprintf('tdm = %g\n',ComboMatrix(10))
fprintf('knm = %g\n',ComboMatrix(11))
end
```

```
slprjdirectory=exist('slprj'); %Checks for existance of rapid accelerator folder
if slprjdirectory ~= 0
rmdir('slprj','s') % This removes the compiled rapid accelerator folder after simulation.
end
close_system('maurer_ap_ml_aug_07',0) % Closes Simulink System to prevent shadowing
%%%% End Iterative Method %%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 'genetic'
%Begin Genetic Algorithm Section%
tic %Start Stop watch
%testcondition=input('Input test condition [eyesopen, visualfeedback, eyesclosed]: ','s'); %ask for test condition, will be GUI eventually.

%%%%%%%%%%%%%%%%%%% Normal Adult Paramters %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%|    Eyes Open     | Visual Feedback |    Eyes Closed    |%
% Parmeter |   AP    |   ML    |   AP    |   ML    |   AP    |   ML    |%
%-----------------------------------------------------------------------%
%    Kp    |  16.7   |  16.7   |  17.5   |  17.1   | 17.55   |  16.5   |%
%-----------------------------------------------------------------------%
%    Ki    |   0.6   |   0.6   |   0.6   |   2.0   |   0.8   |   1.0   |%
%-----------------------------------------------------------------------%
%    Kd    |   6.7   |   9.0   |   6.7   |   9.0   |   6.5   |   8.9   |%
%-----------------------------------------------------------------------%
%    Td    | 0.171   | 0.171   | 0.171   | 0.171   | 0.171   | 0.171   |%
%-----------------------------------------------------------------------%
%    Kn    | 250.0   | 118.0   | 245.0   | 113.0   | 327.0   | 130.0   |%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

geneticuserinput=[RequestedParameters(1),RequestedParameters(2),RequestedParameters(3),RequestedParameters(4),RequestedParameters(5),RequestedParameters(6),...
    RequestedParameters(7),RequestedParameters(8),RequestedParameters(9),RequestedParameters(10)]; %Builds a user input matrix, only used to test for a user input.
geneticuserinputexistance=(sum(geneticuserinput)~=0); %Checks to see if there was a user input (Boolean), any value non zero

switch testcondition %Switches parameters based on test condition.
    case {'eyesopen','Eyesopen','EyesOpen','eo','EO','Eo','eO'} %Define parameters for Eyes Open


        if geneticuserinputexistance==1
%  Reads the 'RequestedParameters' matrix that is initially defined via the front'OpeningGUI.m' panel.
%  This will make a narrower search, +/- 50% of the user requested input.

kpamin= RequestedParameters(1)*0.50;
kpamax= RequestedParameters(1)*1.50;
kiamin= RequestedParameters(2)*0.50;
kiamax= RequestedParameters(2)*1.50;
kdamin= RequestedParameters(3)*0.50;
kdamax= RequestedParameters(3)*1.50;
tdamin= RequestedParameters(4)*0.50;
tdamax= RequestedParameters(4)*1.50;
knamin= RequestedParameters(5)*0.50;
knamax= RequestedParameters(5)*1.50;
kpmmin= RequestedParameters(6)*0.50;
kpmmax= RequestedParameters(6)*1.50;
kimmin= RequestedParameters(7)*0.50;
kimmax= RequestedParameters(7)*1.50;
kdmmin= RequestedParameters(8)*0.50;
kdmmax= RequestedParameters(8)*1.50;
tdmmin= RequestedParameters(9)*0.50;
tdmmax= RequestedParameters(9)*1.50;
knmmin= RequestedParameters(10)*0.50;
```

```
knmmax= RequestedParameters(10)*1.50;

disp('User Requested Parameters Will Be Used');


        else
kpamin= 16.7*0.25; %Builds a wide range search of the normal adult population (Parameters based↙
on "normal adults", min's are -75%, max's are +75%)
kpamax= 16.7*1.75;
kiamin= 0.6*0.25;
kiamax= 0.6*1.75;
kdamin= 6.7*0.25;
kdamax= 6.7*1.75;
tdamin= 0.171*0.25;
tdamax= 0.171*1.75;
knamin= 250*0.25;
knamax= 250*1.75;
kpmmin= 16.7*0.25;
kpmmax= 16.7*1.75;
kimmin= 0.6*0.25;
kimmax= 0.6*1.75;
kdmmin= 9.0*0.25;
kdmmax= 9.0*1.75;
tdmmin= 0.171*0.25;
tdmmax= 0.171*1.75;
knmmin= 118*0.25;
knmmax= 118*1.75;

disp('No User Input - Default Parameters Will Be Used For Search Space');
        end

%ONLY USED FOR TESTING THE ALGORITHM!
    case 'test' %Define parameters for Eyes Open (Parameters based on "normal adults", min's↙
are -75%, max's are +75%)
kpamin= 8.4*.95; % THIS SECTION HAS BEEN CHANGED FOR TESTING PURPOSES!
kpamax= 8.4*1.05; %THESE VALUES ARE CENTERED AROUND OPTIMAL
kiamin= 0.6*.95;
kiamax= 0.6*1.05;
kdamin= 3*.95;
kdamax= 3*1.05;
tdamin= 0.175*.95;
tdamax= 0.175*1.05;
knamin= 260*.95;
knamax= 260*1.05;
kpmmin= 8.7*.95;
kpmmax= 8.7*1.05;
kimmin= 0.2*.95;
kimmax= 0.2*1.05;
kdmmin= 3.0*.95;
kdmmax= 3.0*1.05;
tdmmin= 0.171*.95;
tdmmax= 0.171*1.05;
knmmin= 250*.95;
knmmax= 250*1.05;
%END TEST CASE!

    case {'visualfeedback','VisualFeedback','Visualfeedback','v','vf','VF','vF','Vf','V'} %↙
Define parameters for Visual Feedback

        if geneticuserinputexistance==1
%  Reads the 'RequestedParameters' matrix that is initially defined via the front'OpeningGUI.m'↙
panel.
%  This will make a narrower search, +/- 50% of the user requested input.

kpamin= RequestedParameters(1)*0.50;
kpamax= RequestedParameters(1)*1.50;
kiamin= RequestedParameters(2)*0.50;
kiamax= RequestedParameters(2)*1.50;
```

```
kdamin= RequestedParameters(3)*0.50;
kdamax= RequestedParameters(3)*1.50;
tdamin= RequestedParameters(4)*0.50;
tdamax= RequestedParameters(4)*1.50;
knamin= RequestedParameters(5)*0.50;
knamax= RequestedParameters(5)*1.50;
kpmmin= RequestedParameters(6)*0.50;
kpmmax= RequestedParameters(6)*1.50;
kimmin= RequestedParameters(7)*0.50;
kimmax= RequestedParameters(7)*1.50;
kdmmin= RequestedParameters(8)*0.50;
kdmmax= RequestedParameters(8)*1.50;
tdmmin= RequestedParameters(9)*0.50;
tdmmax= RequestedParameters(9)*1.50;
knmmin= RequestedParameters(10)*0.50;
knmmax= RequestedParameters(10)*1.50;
disp('User Requested Parameters Will Be Used');

        else

kpamin= 17.5*0.25;  %Builds a wide range search of the normal adult population (Parameters↙
based on "normal adults", min's are -75%, max's are +75%)
kpamax= 17.5*1.75;
kiamin= 0.6*0.25;
kiamax= 0.6*1.75;
kdamin= 6.7*0.25;
kdamax= 6.7*1.75;
tdamin= 0.171*0.25;
tdamax= 0.171*1.75;
knamin= 245*0.25;
knamax= 245*1.75;
kpmmin= 17.1*0.25;
kpmmax= 17.1*1.75;
kimmin= 2.0*0.25;
kimmax= 2.0*1.75;
kdmmin= 9.0*0.25;
kdmmax= 9.0*1.75;
tdmmin= 0.171*0.25;
tdmmax= 0.171*1.75;
knmmin= 113*0.25;
knmmax= 113*1.75;

disp('No User Input - Default Parameters Will Be Used For Search Space');

        end


    case {'eyesclosed','Eyesclosed','EyesClosed','ec','Ec','eC','EC'} %Define parameters for↙
Eyes Closed

        if geneticuserinputexistance==1
%  Reads the 'RequestedParameters' matrix that is initially defined via the front'OpeningGUI.m'↙
panel.
%  This will make a narrower search, +/- 50% of the user requested input.

kpamin= RequestedParameters(1)*0.50;
kpamax= RequestedParameters(1)*1.50;
kiamin= RequestedParameters(2)*0.50;
kiamax= RequestedParameters(2)*1.50;
kdamin= RequestedParameters(3)*0.50;
kdamax= RequestedParameters(3)*1.50;
tdamin= RequestedParameters(4)*0.50;
tdamax= RequestedParameters(4)*1.50;
knamin= RequestedParameters(5)*0.50;
knamax= RequestedParameters(5)*1.50;
kpmmin= RequestedParameters(6)*0.50;
kpmmax= RequestedParameters(6)*1.50;
kimmin= RequestedParameters(7)*0.50;
```

```
kimmax= RequestedParameters(7)*1.50;
kdmmin= RequestedParameters(8)*0.50;
kdmmax= RequestedParameters(8)*1.50;
tdmmin= RequestedParameters(9)*0.50;
tdmmax= RequestedParameters(9)*1.50;
knmmin= RequestedParameters(10)*0.50;
knmmax= RequestedParameters(10)*1.50;
disp('User Requested Parameters Will Be Used');
        else

kpamin= 17.55*0.25;     %Builds a wide range search of the normal adult population (Parameters↙
based on "normal adults", min's are -75%, max's are +75%)
kpamax= 17.55*1.75;
kiamin= 0.8*0.25;
kiamax= 0.8*1.75;
kdamin= 6.5*0.25;
kdamax= 6.5*1.75;
tdamin= 0.171*0.25;
tdamax= 0.171*1.75;
knamin= 327*0.25;
knamax= 327*1.75;
kpmmin= 16.5*0.25;
kpmmax= 16.5*1.75;
kimmin= 1.0*0.25;
kimmax= 1.0*1.75;
kdmmin= 8.9*0.25;
kdmmax= 8.9*1.75;
tdmmin= 0.171*0.25;
tdmmax= 0.171*1.75;
knmmin= 130*0.25;
knmmax= 130*1.75;

disp('No User Input - Default Parameters Will Be Used For Search Space');
        end

  otherwise
               disp('Test Condition Not Valid - End Simulation')
               diary off
              return

end
%%%%%%%Initialized Parameters%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
kpadiff=kpamax-kpamin; %Create value for min/max span
kiadiff=kiamax-kiamin;
kdadiff=kdamax-kdamin;
tdadiff=tdamax-tdamin;
knadiff=knamax-knamin;
kpmdiff=kpmmax-kpmmin;
kimdiff=kimmax-kimmin;
kdmdiff=kdmmax-kdmmin;
tdmdiff=tdmmax-tdmmin;
knmdiff=knmmax-knmmin;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


for restart=1:(reruns+1) %Allows Algorithm to ru-run after a number of generations

clear fitnessout %resets variables.
clear kpaout
clear kiaout
clear kdaout
clear tdaout
clear knaout
clear kpmout
clear kimout
clear kdmout
```

```
clear tdmout
clear knmout
clear population

%populationsize=input('Population size: '); %This is overridden by GUI
%numberofgenerations=input('Number of Generations: '); %Also in GUI
fitness=zeros(populationsize,1); %Set all fitness functions to zero.

population=[kpamin+(rand(populationsize,1)*kpadiff),kiamin+(rand(populationsize,1)*kiadiff),↙
kdamin+(rand(populationsize,1)*kdadiff),tdamin+(rand(populationsize,1)*tdadiff),knamin+(rand↙
(populationsize,...
    1)*knadiff),kpmmin+(rand(populationsize,1)*kpmdiff),kimmin+(rand(populationsize,1)↙
*kimdiff),kdmmin+(rand(populationsize,1)*kdmdiff),tdmmin+(rand(populationsize,1)*tdmdiff),...
    knmmin+(rand(populationsize,1)*knmdiff),fitness]; %Create Initial Population
if javaon==1
k = waitbar(0,'Algorithm Running, Please wait...'); %Opens waitbar
end

cmap = lines(reruns+1);%Build a colormap for plotting

for y=1:numberofgenerations,

%%%%%------------------------------------------------%%%%
% This section makes sure to pick valid parents, not to choose parents
% which resulted in an error, unless it's the first generation, then it's a
% random choice of the first 10


validpopulationmembers=population(:,11)<999;
validpopulationnumber=sum(validpopulationmembers);
if validpopulationnumber > 10
    validpopulationnumber=10;
end

if y==1
    validpopulationnumber=10;
end
%%%%%%------------------------------------------------%%%%%


parents=randi([1,validpopulationnumber],[2,1]);%Randomly Select 2 Parents to make child from↙
top 10 parents

child1=[(population(parents(1),1:5)),(population(parents(2),6:10)),0]; %Creates First Child
child2=[(population(parents(2),1:5)),(population(parents(1),6:10)),0]; %Creates Second Child
mutantcharacterisitc=randi([1,10]); %Determines which characteristic to modify
mutantvalue1=child1(mutantcharacterisitc)*rand;%Decreases mutant 1 value based on child 1
mutantvalue2=child2(mutantcharacterisitc)*rand;%Decreases mutant 2 value based on child 2
mutantvalue3=child1(mutantcharacterisitc)*(1+rand);%Increases mutant 3 value based on child 1
mutantvalue4=child2(mutantcharacterisitc)*(1+rand);%Increases mutant 4 value based on child 2
mutant1=child1;%Gives the mutant base characteristics
mutant2=child2;
mutant3=child1;
mutant4=child2;
mutant1(mutantcharacterisitc)=mutantvalue1; %Creates a mutant child with mutant characteristic↙
value multiplied by a random value 0->1;
mutant2(mutantcharacterisitc)=mutantvalue2; %Creates a mutant child with mutant characteristic↙
value multiplied by a random value 0->1;
mutant3(mutantcharacterisitc)=mutantvalue3; %Creates a mutant child with mutant characteristic↙
value multiplied by a random value 1->2;
mutant4(mutantcharacterisitc)=mutantvalue4; %Creates a mutant child with mutant characteristic↙
value multiplied by a random value 1->2;

switch mutantcharacterisitc %Check to m make sure mutant characteristic is still within↙
original bounds
    case 1
        if(((mutantvalue1>kpamin)&&(mutantvalue1<kpamax))==0);
        mutant1=[];
```

```
    end

    if(((mutantvalue2>kpamin)&&(mutantvalue2<kpamax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>kpamin)&&(mutantvalue3<kpamax))==0);
    mutant3=[];
    end

    if(((mutantvalue4>kpamin)&&(mutantvalue4<kpamax))==0);
    mutant4=[];
    end

case 2
    if(((mutantvalue1>kiamin)&&(mutantvalue1<kiamax))==0);
    mutant1=[];
    end

    if(((mutantvalue2>kiamin)&&(mutantvalue2<kiamax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>kiamin)&&(mutantvalue3<kiamax))==0);
    mutant3=[];
    end

    if(((mutantvalue4>kiamin)&&(mutantvalue4<kiamax))==0);
    mutant4=[];
    end

case 3
    if(((mutantvalue1>kdamin)&&(mutantvalue1<kdamax))==0);
    mutant1=[];
    end

    if(((mutantvalue2>kdamin)&&(mutantvalue2<kdamax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>kdamin)&&(mutantvalue3<kdamax))==0);
    mutant3=[];
    end

    if(((mutantvalue4>kdamin)&&(mutantvalue4<kdamax))==0);
    mutant4=[];
    end

case 4

    if(((mutantvalue1>tdamin)&&(mutantvalue1<tdamax))==0);
    mutant1=[];
    end

    if(((mutantvalue2>tdamin)&&(mutantvalue2<tdamax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>tdamin)&&(mutantvalue3<tdamax))==0);
    mutant3=[];
    end

    if(((mutantvalue4>tdamin)&&(mutantvalue4<tdamax))==0);
    mutant4=[];
    end

case 5
```

```
    if(((mutantvalue1>knamin)&&(mutantvalue1<knamax))==0);
    mutant1=[];
    end

    if(((mutantvalue2>knamin)&&(mutantvalue2<knamax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>knamin)&&(mutantvalue3<knamax))==0);
    mutant3=[];
    end

    if(((mutantvalue4>knamin)&&(mutantvalue4<knamax))==0);
    mutant4=[];
    end

case 6
    if(((mutantvalue1>kpmmin)&&(mutantvalue1<kpmmax))==0);
    mutant1=[];
    end

    if(((mutantvalue2>kpmmin)&&(mutantvalue2<kpmmax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>kpmmin)&&(mutantvalue3<kpmmax))==0);
    mutant3=[];
    end

    if(((mutantvalue4>kpmmin)&&(mutantvalue4<kpmmax))==0);
    mutant4=[];
    end

case 7
    if(((mutantvalue1>kimmin)&&(mutantvalue1<kimmax))==0);
    mutant1=[];
    end

    if(((mutantvalue2>kimmin)&&(mutantvalue2<kimmax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>kimmin)&&(mutantvalue3<kimmax))==0);
    mutant3=[];
    end
    if(((mutantvalue4>kimmin)&&(mutantvalue4<kimmax))==0);
    mutant4=[];
    end

case 8
    if(((mutantvalue1>kdmmin)&&(mutantvalue1<kdmmax))==0);
    mutant1=[];
    end

    if(((mutantvalue2>kdmmin)&&(mutantvalue2<kdmmax))==0);
    mutant2=[];
    end

    if(((mutantvalue3>kdmmin)&&(mutantvalue3<kdmmax))==0);
    mutant3=[];
    end

    if(((mutantvalue4>kdmmin)&&(mutantvalue4<kdmmax))==0);
    mutant4=[];
    end

case 9
    if(((mutantvalue1>tdmmin)&&(mutantvalue1<tdmmax))==0);
```

```
        mutant1=[];
        end

        if(((mutantvalue2>tdmmin)&&(mutantvalue2<tdmmax))==0);
        mutant2=[];
        end

        if(((mutantvalue3>tdmmin)&&(mutantvalue3<tdmmax))==0);
        mutant3=[];
        end

        if(((mutantvalue4>tdmmin)&&(mutantvalue4<tdmmax))==0);
        mutant4=[];
        end

    case 10
        if(((mutantvalue1>knmmin)&&(mutantvalue1<knmmax))==0);
        mutant1=[];
        end

        if(((mutantvalue2>knmmin)&&(mutantvalue2<knmmax))==0);
        mutant2=[];
        end

        if(((mutantvalue3>knmmin)&&(mutantvalue3<knmmax))==0);
        mutant3=[];
        end

        if(((mutantvalue4>knmmin)&&(mutantvalue4<knmmax))==0);
        mutant4=[];
        end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

changedpopulation=[child1;mutant1;mutant3;child2;mutant2;mutant4]; %Creates  matrix of changed↙
values.
%Child1, Mutant of child1, second mutant of child 1
%Repeat for child 2.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('\nGeneration %g\n',y)% Show Generation


if y==1 %For first generation
% Finds fitness of population

disp('Building Initial Population');
population=[population;changedpopulation];% Appends population
%size(population)
for z=1:size(population,1),


kpa=population(z,1); %Defines each variable in population matrix.
kia=population(z,2);
kda=population(z,3);
tda=population(z,4);
kna=population(z,5);
kpm=population(z,6);
kim=population(z,7);
kdm=population(z,8);
tdm=population(z,9);
knm=population(z,10);
fitness=population(z,11);

try
load('StaticVariables.mat', 'eo', 'eof', 'ec', 'cpdip', 'cphem', 'J', 'm', 'h', 'g', 'noiseap',↙
'noiseml', 'means', 'sd', 'means_array', 'meanseof', 'sdeof', 'means_arrayeof', 'meansec',↙
```

```
'sdec', 'means_arrayec', 'ranges', 'ranges2', 'ranges3');  % Reads required simulation elements from 'StaticVariables.mat' - an always-constant file that is never written to.  This read occurs at the start of each iteration as well.

%=========================================================================%
%============\/ ORIGINAL CALC \/============%
%==========%

if eof==1
    means=meanseof;
    sd=sdeof;
    means_array=means_arrayeof;
end

if ec==1
    means=meansec;
    sd=sdec;
    means_array=means_arrayec;
end


 n1=1;   n2=15;
 m1=1;   m2=2;
 %noiseml=noiseap; just to check
  for i=1:7

    sim('maurer_ap_ml_aug_07',451)

    ap=cop1(tetarad_ap(:,2)); ml=cop1(tetarad_ml(:,2));  %Convert from teta to cop
    results=zeros(38,15);

    results1=calculos_simulation_june_07(ap(1:3000),ml(1:3000),'First combination');
    results(:,1)=results1;
    r1=(abs(results1-means)>sd);


    results2=calculos_simulation_june_07(ap(3001:6000),ml(3001:6000),'2 combination');
    r2=(abs(results2-means)>sd);
    results(:,2)=results2;

    results3=calculos_simulation_june_07(ap(6001:9000),ml(6001:9000),'3 combination');
    r3=(abs(results3-means)>sd);
    results(:,3)=results3;

    results4=calculos_simulation_june_07(ap(9001:12000),ml(9001:12000),'4 combination');
    r4=(abs(results4-means)>sd);
    results(:,4)=results4;

    results5=calculos_simulation_june_07(ap(12001:15000),ml(12001:15000),'5 combination');
    r5=(abs(results5-means)>sd);
    results(:,5)=results5;

    results6=calculos_simulation_june_07(ap(15001:18000),ml(15001:18000),'6 combination');
    r6=(abs(results6-means)>sd);
    results(:,6)=results6;

    results7=calculos_simulation_june_07(ap(18001:21000),ml(18001:21000),'7 combination');
    r7=(abs(results7-means)>sd);
    results(:,7)=results7;

    results8=calculos_simulation_june_07(ap(21001:24000),ml(21001:24000),'8 combination');
    r8=(abs(results8-means)>sd);
    results(:,8)=results8;

    results9=calculos_simulation_june_07(ap(24001:27000),ml(24001:27000),'9 combination');
    r9=(abs(results9-means)>sd);
    results(:,9)=results9;
```

```
    results10=calculos_simulation_june_07(ap(27001:30000),ml(27001:30000),'10 combination');
    r10=(abs(results10-means)>sd);
    results(:,10)=results10;

    results11=calculos_simulation_june_07(ap(30001:33000),ml(30001:33000),'11 combination');
    r11=(abs(results11-means)>sd);
    results(:,11)=results11;

    results12=calculos_simulation_june_07(ap(33001:36000),ml(33001:36000),'12 combination');
    r12=(abs(results12-means)>sd);
    results(:,12)=results12;

    results13=calculos_simulation_june_07(ap(36001:39000),ml(36001:39000),'13 combination');
    r13=(abs(results13-means)>sd);
    results(:,13)=results13;

    results14=calculos_simulation_june_07(ap(39001:42000),ml(39001:42000),'14 combination');
    r14=(abs(results14-means)>sd);
    results(:,14)=results14;

    results15=calculos_simulation_june_07(ap(42001:45000),ml(42001:45000),'15 combination');
    r15=(abs(results14-means)>sd);
    results(:,15)=results15;

    meanresults=mean(results,2);
    rmean=(abs(meanresults-means)>sd);


    simulation(:,n1:n2)=results;
    n1=n1+15;
    n2=n2+15;


    res(:,m1:m2)=[meanresults,rmean];
    res1(:,i)=rmean;

    m1=m1+2;
    m2=m2+2;

    noiseap=noiseap-1;  %initial noise seed in ap at 23340
    %noiseml=noiseap;
    noiseml=noiseml+1;  %initial noise seed in ml at 23343
end



n1=1;
n2=3;
for i=1:35
    simulationsmeans(:,i)=mean(simulation(:,n1:n2),2);
    n1=n1+3;
    n2=n2+3;
end

statsarray=simulationsmeans(:,1:17); %EDIT HERE to proper array size.  (:,1:n) ,where 'n' is↙
the array size to edit!  This value is probably set at '17' or '31' now.
overallmean=mean(statsarray,2);
statsarray=statsarray';

%display('end')

n1=1;
n2=2;
n3=3;
for i=1:35
    array(:,n1)=means_array(:,i);
    array(:,n2)=statsarray(:,i);
    array(:,n3)=zeros(17,1); %EDIT HERE to proper array size.  (n,1) ,where 'n' is the array↙
```

```
size to edit!  This value is probably set at '17' or '31' now.
    n1=n1+3;
    n2=n2+3;
    n3=n3+3;
end

highervalue=max(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.  Also,↙
'gives me the maximum value in a given metric'
lowervalue=min(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.

arr=[highervalue lowervalue];
comparison1=ranges3(:,1)-arr(:,1);
comparison2=arr(:,2)-ranges3(:,2);
[h,p]=ttest(means_array,statsarray,0.005);
arr1=[overallmean comparison1 comparison2 h'];
%==========%
%===========/\ ORIGINAL CALC /\===========%
%=========================================================================%

TotalMismatchesOverall = sum(sum(res(1:35,2:2:14)));
fprintf('Population Member %g Mismatches: %g\n',z,TotalMismatchesOverall)

fitnessout(z,1)=TotalMismatchesOverall;
kpaout(z,1)=kpa;
kiaout(z,1)=kia;
kdaout(z,1)=kda;
tdaout(z,1)=tda;
knaout(z,1)=kna;
kpmout(z,1)=kpm;
kimout(z,1)=kim;
kdmout(z,1)=kdm;
tdmout(z,1)=tdm;
knmout(z,1)=knm;



if (TotalMismatchesOverall<=InputConstraint)
   fprintf('\nGoal Mismatches Reached.\n')
   population=[kpaout, kiaout, kdaout, tdaout, knaout, kpmout, kimout, kdmout,tdmout,knmout,↙
fitnessout];%Rebuilt population matrix.
   population=unique(population,'rows'); % Make sure population is unique
    population=sortrows(population,11); %Creates a sorted matrix based on fitness.
    Bestfit=[kpa, kia, kda, tda,kna,kpm,kim,kdm,tdm,knm];
    if javaon==1
        close(k) %Closes waitbar
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%BUILD OUTPUT FILE
resultsdirectory=exist('Simulation_Results','dir');
if resultsdirectory~=7
    mkdir('Simulation_Results')
end

%Save Plots
if javaon==1
plotdirectory=exist('Plots','dir');
if plotdirectory~=7
    mkdir('Plots')
end
    plotfilename=sprintf('Plots/Plot_%g_%g_%g_%g_%g',date(1),date(2),date(3),date(4),date(5));
    saveas(gcf,plotfilename,'png')
end


simtime=toc;

filename=sprintf('Simulation_Results/Simulation_Results_%g_%g_%g_%g_%g.txt',date(1),date(2),↙
```

```
date(3),date(4),date(5));
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf('Printing to a file: %s \n',filename);
u = fopen(filename,'w'); %open output file
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf(u,'Filename - %s\n',filename);
fprintf('Simulation Completed in %g Generation(s) --- Simulation Time: %g Seconds\n',y,↙
simtime);
fprintf(u,'Simulation Completed in %g Generation(s) --- Simulation Time: %g Seconds\n',y,↙
simtime);
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm↙
\t\t Mismatches\n');
fprintf(u,'kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm↙
\t\t Mismatches\n');
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('Best Fit\n');
fprintf(u,'Best Fit\n');
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('Entire Population\n');
fprintf(u,'Entire Population\n');

for j=1:size(population,1)
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population↙
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),↙
population(j,10),population(j,11));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population↙
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),↙
population(j,10),population(j,11));
end

fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fclose(u);
%close output file
diary off %Ends Log File
close_system('maurer_ap_ml_aug_07',0) % Closes Simulink System to prevent shadowing
%%%%%%%%%%%%%%%END BUILD OUTPUT FILE
```

```
slprjdirectory=exist('slprj'); %Checks for existance of rapid accelerator folder
if slprjdirectory ~= 0
rmdir('slprj','s') % This removes the compiled rapid accelerator folder after simulation.
end
    return


end

continue


catch
fprintf('Population Member %g Mismatches: Error\n',z)
fitnessout(z,1)=999;

kpaout(z,1)=kpa;
kiaout(z,1)=kia;
kdaout(z,1)=kda;
tdaout(z,1)=tda;
knaout(z,1)=kna;
kpmout(z,1)=kpm;
kimout(z,1)=kim;
kdmout(z,1)=kdm;
tdmout(z,1)=tdm;
knmout(z,1)=knm;


 end



end


population=[kpaout, kiaout, kdaout, tdaout, knaout, kpmout, kimout, kdmout,tdmout,knmout,↙
fitnessout];%Rebuilt population matrix.
population=unique(population,'rows'); % Make sure population is unique
population=sortrows(population,11); %Creates a sorted matrix based on fitness.
population=population(1:populationsize,:);%Crops matrix to initial population size based on↙
fitness.

if sum(population(:,11))>=(999*(size(population,1)-1))  %Error Catching For Initial Population.
    fprintf('\nInitial Population Did Not Find At Least Two Viable Parents - End Simulation\n')
    fprintf('\tTry Again or Use a Larger Population\n')
    diary off

    if javaon==1
        close(k) %Closes progress bar
    end
    return

end

firstroundpopulation=population; %Makes a variable for the first population, keeps track of↙
values already evaluated.
bestfitness=population(1,11); %Choose best fitness of the generation.
fprintf('Best Fitness of Generation %g: %g\n',y,bestfitness) %Display the best fitness of the↙
generation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if javaon==1 %Checks if java is available
waitbar(y/numberofgenerations,k) %Performs Waitbar Calculation



plotfitness(restart,y)=population(1,11); %Create vector of the best fitness value per↙
```

```
generation
hold on;
hplot(restart)=plot(plotfitness(restart,1:y),'--s','Color',cmap(restart,:),'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',cmap(restart,:),...
                'MarkerSize',5); %Plot the best fitness value per generation

%axis([0 y 0 max(plotfitness)+1])
xlabel('Generation')
ylabel('Fitness (Number of Mismatches)')
title('Best Fitness vs. Generation')

end

else

clear fitnessout %resets variables.
clear kpaout
clear kiaout
clear kdaout
clear tdaout
clear knaout
clear kpmout
clear kimout
clear kdmout
clear tdmout
clear knmout

for z=1:size(changedpopulation,1),

kpa=changedpopulation(z,1); %Defines each variable in population matrix.
kia=changedpopulation(z,2);
kda=changedpopulation(z,3);
tda=changedpopulation(z,4);
kna=changedpopulation(z,5);
kpm=changedpopulation(z,6);
kim=changedpopulation(z,7);
kdm=changedpopulation(z,8);
tdm=changedpopulation(z,9);
knm=changedpopulation(z,10);
fitness=changedpopulation(z,11);
try
load('StaticVariables.mat', 'eo', 'eof', 'ec', 'cpdip', 'cphem', 'J', 'm', 'h', 'g', 'noiseap',↙
'noiseml', 'means', 'sd', 'means_array', 'meanseof', 'sdeof', 'means_arrayeof', 'meansec',↙
'sdec', 'means_arrayec', 'ranges', 'ranges2', 'ranges3');  % Reads required simulation elements↙
from 'StaticVariables.mat' - an always-constant file that is never written to.  This read↙
occurs at the start of each iteration as well.

%=========================================================================%
%============\/ ORIGINAL CALC \/============%
%==========%

if eof==1
    means=meanseof;
    sd=sdeof;
    means_array=means_arrayeof;
end

if ec==1
    means=meansec;
    sd=sdec;
    means_array=means_arrayec;
end


 n1=1;    n2=15;
 m1=1;    m2=2;
 %noiseml=noiseap; just to check
```

```
for i=1:7

  sim('maurer_ap_ml_aug_07',451)

  ap=cop1(tetarad_ap(:,2)); ml=cop1(tetarad_ml(:,2));  %Convert from teta to cop
  results=zeros(38,15);

  results1=calculos_simulation_june_07(ap(1:3000),ml(1:3000),'First combination');
  results(:,1)=results1;
  r1=(abs(results1-means)>sd);


  results2=calculos_simulation_june_07(ap(3001:6000),ml(3001:6000),'2 combination');
  r2=(abs(results2-means)>sd);
  results(:,2)=results2;

  results3=calculos_simulation_june_07(ap(6001:9000),ml(6001:9000),'3 combination');
  r3=(abs(results3-means)>sd);
  results(:,3)=results3;

  results4=calculos_simulation_june_07(ap(9001:12000),ml(9001:12000),'4 combination');
  r4=(abs(results4-means)>sd);
  results(:,4)=results4;

  results5=calculos_simulation_june_07(ap(12001:15000),ml(12001:15000),'5 combination');
  r5=(abs(results5-means)>sd);
  results(:,5)=results5;

  results6=calculos_simulation_june_07(ap(15001:18000),ml(15001:18000),'6 combination');
  r6=(abs(results6-means)>sd);
  results(:,6)=results6;

  results7=calculos_simulation_june_07(ap(18001:21000),ml(18001:21000),'7 combination');
  r7=(abs(results7-means)>sd);
  results(:,7)=results7;

  results8=calculos_simulation_june_07(ap(21001:24000),ml(21001:24000),'8 combination');
  r8=(abs(results8-means)>sd);
  results(:,8)=results8;

  results9=calculos_simulation_june_07(ap(24001:27000),ml(24001:27000),'9 combination');
  r9=(abs(results9-means)>sd);
  results(:,9)=results9;

  results10=calculos_simulation_june_07(ap(27001:30000),ml(27001:30000),'10 combination');
  r10=(abs(results10-means)>sd);
  results(:,10)=results10;

  results11=calculos_simulation_june_07(ap(30001:33000),ml(30001:33000),'11 combination');
  r11=(abs(results11-means)>sd);
  results(:,11)=results11;

  results12=calculos_simulation_june_07(ap(33001:36000),ml(33001:36000),'12 combination');
  r12=(abs(results12-means)>sd);
  results(:,12)=results12;

  results13=calculos_simulation_june_07(ap(36001:39000),ml(36001:39000),'13 combination');
  r13=(abs(results13-means)>sd);
  results(:,13)=results13;

  results14=calculos_simulation_june_07(ap(39001:42000),ml(39001:42000),'14 combination');
  r14=(abs(results14-means)>sd);
  results(:,14)=results14;

  results15=calculos_simulation_june_07(ap(42001:45000),ml(42001:45000),'15 combination');
  r15=(abs(results14-means)>sd);
  results(:,15)=results15;
```

```
    meanresults=mean(results,2);
    rmean=(abs(meanresults-means)>sd);


    simulation(:,n1:n2)=results;
    n1=n1+15;
    n2=n2+15;


    res(:,m1:m2)=[meanresults,rmean];
    res1(:,i)=rmean;

    m1=m1+2;
    m2=m2+2;

    noiseap=noiseap-1;  %initial noise seed in ap at 23340
    %noiseml=noiseap;
    noiseml=noiseml+1;  %initial noise seed in ml at 23343
end



n1=1;
n2=3;
for i=1:35
    simulationsmeans(:,i)=mean(simulation(:,n1:n2),2);
    n1=n1+3;
    n2=n2+3;
end

statsarray=simulationsmeans(:,1:17); %EDIT HERE to proper array size.  (:,1:n) ,where 'n' is↙
the array size to edit!  This value is probably set at '17' or '31' now.
overallmean=mean(statsarray,2);
statsarray=statsarray';

%display('end')

n1=1;
n2=2;
n3=3;
for i=1:35
    array(:,n1)=means_array(:,i);
    array(:,n2)=statsarray(:,i);
    array(:,n3)=zeros(17,1); %EDIT HERE to proper array size.  (n,1) ,where 'n' is the array↙
size to edit!  This value is probably set at '17' or '31' now.
    n1=n1+3;
    n2=n2+3;
    n3=n3+3;
end

highervalue=max(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.  Also,↙
'gives me the maximum value in a given metric'
lowervalue=min(simulationsmeans(:,1:17),[],2); %EDIT HERE to proper array size.  (:,1:n),[],2)↙
,where 'n' is the array size to edit!  This value is probably set at '17' or '31' now.

arr=[highervalue lowervalue];
comparison1=ranges3(:,1)-arr(:,1);
comparison2=arr(:,2)-ranges3(:,2);
[h,p]=ttest(means_array,statsarray,0.005);
arr1=[overallmean comparison1 comparison2 h'];
%==========%
%===========/\ ORIGINAL CALC /\===========%
%=========================================================================%

TotalMismatchesOverall = sum(sum(res(1:35,2:2:14)));
fprintf('New Population Member %g Mismatches: %g\n',z,TotalMismatchesOverall)
```

```
fitnessout(z,1)=TotalMismatchesOverall;
kpaout(z,1)=kpa;
kiaout(z,1)=kia;
kdaout(z,1)=kda;
tdaout(z,1)=tda;
knaout(z,1)=kna;
kpmout(z,1)=kpm;
kimout(z,1)=kim;
kdmout(z,1)=kdm;
tdmout(z,1)=tdm;
knmout(z,1)=knm;

if (TotalMismatchesOverall<=InputConstraint)
   fprintf('\nGoal Mismatches Reached.\n')
population=[kpaout, kiaout, kdaout, tdaout, knaout, kpmout, kimout, kdmout,tdmout,knmout,↙
fitnessout;population];%Rebuilt population matrix.
population=unique(population,'rows'); % Make sure population is unique
population=sortrows(population,11); %Creates a sorted matrix based on fitness.
population=population(1:populationsize,:);%Crops matrix to initial population size based on↙
fitness.
    Bestfit=[kpa, kia, kda, tda,kna,kpm,kim,kdm,tdm,knm];
    toc %ends Stop Watch.
    if javaon==1
plotfitness(restart,y)=population(1,11); %Create vector of the best fitness value per↙
generation
hold on;
hplot(restart)=plot(plotfitness(restart,1:y),'--s','Color',cmap(restart,:),'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',cmap(restart,:),...
                'MarkerSize',5); %Plot the best fitness value per generation

%axis([1 y 0 max(plotfitness)+1])
xlabel('Generation')
ylabel('Fitness (Number of Mismatches)')
title('Best Fitness vs. Generation')

     close(k) %Closes waitbar
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%BUILD OUTPUT FILE
    resultsdirectory=exist('Simulation_Results','dir');
if resultsdirectory~=7
    mkdir('Simulation_Results')
end

%Save Plots
if javaon==1
plotdirectory=exist('Plots','dir');
if plotdirectory~=7
    mkdir('Plots')
end
    plotfilename=sprintf('Plots/Plot_%g_%g_%g_%g_%g',date(1),date(2),date(3),date(4),date(5));
    saveas(gcf,plotfilename,'png')
end

simtime=toc;

filename=sprintf('Simulation_Results/Simulation_Results_%g_%g_%g_%g_%g.txt',date(1),date(2),↙
date(3),date(4),date(5));
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf('Printing to a file: %s \n',filename);
u = fopen(filename,'w'); %open output file
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf(u,'Filename - %s\n',filename);
fprintf('Simulation Completed in %g Generation(s) --- Simulation Time: %g Seconds\n',y,↙
```

```
simtime);
fprintf(u,'Simulation Completed in %g Generation(s) --- Simulation Time: %g Seconds\n',y,
simtime);
fprintf
('=============================================================================================
==========================================================================\n');
fprintf
(u,'=============================================================================================
==========================================================================\n');
fprintf('kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm
\t\t Mismatches\n');
fprintf(u,'kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm
\t\t Mismatches\n');
fprintf
('=============================================================================================
==========================================================================\n');
fprintf
(u,'=============================================================================================
==========================================================================\n');
fprintf('Best Fit\n');
fprintf(u,'Best Fit\n');
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf
('=============================================================================================
==========================================================================\n');
fprintf
(u,'=============================================================================================
==========================================================================\n');
fprintf('Entire Population\n');
fprintf(u,'Entire Population\n');

for j=1:size(population,1)
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),
population(j,10),population(j,11));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),
population(j,10),population(j,11));
end

fprintf
('=============================================================================================
==========================================================================\n');
fprintf
(u,'=============================================================================================
==========================================================================\n');
fclose(u);
%close output file
diary off %Ends Log File
close_system('maurer_ap_ml_aug_07',0) % Closes Simulink System to prevent shadowing

%%%%%%%%%%%%%%%END BUILD OUTPUT FILE
slprjdirectory=exist('slprj'); %Checks for existance of rapid accelerator folder
if slprjdirectory ~= 0
rmdir('slprj','s') % This removes the compiled rapid accelerator folder after simulation.
end
    return
end


continue

catch
```

```
fprintf('New Population Member %g Mismatches: Error\n',z)
fitnessout(z,1)=999;

kpaout(z,1)=kpa;
kiaout(z,1)=kia;
kdaout(z,1)=kda;
tdaout(z,1)=tda;
knaout(z,1)=kna;
kpmout(z,1)=kpm;
kimout(z,1)=kim;
kdmout(z,1)=kdm;
tdmout(z,1)=tdm;
knmout(z,1)=knm;


 end

if (TotalMismatchesOverall<=InputConstraint)
   fprintf('\nGoal Mismatches Reached.\n')
population=[kpaout, kiaout, kdaout, tdaout, knaout, kpmout, kimout, kdmout,tdmout,knmout,↙
fitnessout;population];%Rebuilt population matrix.
population=unique(population,'rows'); % Make sure population is unique
population=sortrows(population,11); %Creates a sorted matrix based on fitness.
population=population(1:populationsize,:);%Crops matrix to initial population size based on↙
fitness.
    Bestfit=[kpa, kia, kda, tda,kna,kpm,kim,kdm,tdm,knm];
    if javaon==1

plotfitness(restart,y)=population(1,11); %Create vector of the best fitness value per↙
generation
hold on;
hplot(restart)=plot(plotfitness(restart,1:y),'--s','Color',cmap(restart,:),'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',cmap(restart,:),...
                'MarkerSize',5); %Plot the best fitness value per generation

%axis([1 y 0 max(plotfitness)+1])
xlabel('Generation')
ylabel('Fitness (Number of Mismatches)')
title('Best Fitness vs. Generation')

    close(k) %Closes waitbar
    end

     simtime=toc; %ends Stop Watch
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%BUILD OUTPUT FILE
     resultsdirectory=exist('Simulation_Results','dir');
if resultsdirectory~=7
    mkdir('Simulation_Results')
end
%Save Plots
if javaon==1
plotdirectory=exist('Plots','dir');
if plotdirectory~=7
    mkdir('Plots')
end
    plotfilename=sprintf('Plots/Plot_%g_%g_%g_%g_%g',date(1),date(2),date(3),date(4),date(5));
    saveas(gcf,plotfilename,'png')
end

filename=sprintf('Simulation_Results/Simulation_Results_%g_%g_%g_%g_%g.txt',date(1),date(2),↙
date(3),date(4),date(5));
fprintf↙
('================================================================================================↙
==========================================================================\n');
fprintf('Printing to a file: %s \n',filename);
u = fopen(filename,'w'); %open output file
fprintf↙
```

```
('=============================================================================================↙
==========================================================================\n');
fprintf(u,'Filename - %s\n',filename);
fprintf('Simulation Completed in %g Generation(s) --- Simulation Time: %g Seconds\n',y,↙
simtime);
fprintf(u,'Simulation Completed in %g Generation(s) --- Simulation Time:  %g Seconds\n',y,↙
simtime);
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm↙
\t\t Mismatches\n');
fprintf(u,'kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm↙
\t\t Mismatches\n');
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('Best Fit\n');
fprintf(u,'Best Fit\n');
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('Entire Population\n');
fprintf(u,'Entire Population\n');

for j=1:size(population,1)
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population↙
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),↙
population(j,10),population(j,11));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population↙
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),↙
population(j,10),population(j,11));
end

fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fclose(u);
%close output file
close_system('maurer_ap_ml_aug_07',0) % Closes Simulink System to prevent shadowing
diary off %Ends Log File
%%%%%%%%%%%%%%%END BUILD OUTPUT FILE
slprjdirectory=exist('slprj'); %Checks for existance of rapid accelerator folder
if slprjdirectory ~= 0
rmdir('slprj','s') % This removes the compiled rapid accelerator folder after simulation.
end
    return
end
```

```
end


population=[kpaout, kiaout, kdaout, tdaout, knaout, kpmout, kimout, kdmout,tdmout,knmout,↙
fitnessout;population];%Rebuilt population matrix.
population=unique(population,'rows'); % Make sure population is unique

population=sortrows(population,11); %Creates a sorted matrix based on fitness.
population=population(1:populationsize,:);%Crops matrix to initial population size based on↙
fitness.
bestfitness=population(1,11); %Choose best fitness of the generation.
fprintf('Best Fitness of Generation %g: %g\n',y,bestfitness) %Display the best fitness of the↙
generation.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if javaon==1 %Checks if java is available
waitbar(y/numberofgenerations,k) %Performs Waitbar Calculation



plotfitness(restart,y)=population(1,11); %Create vector of the best fitness value per↙
generation
hold on;
hplot(restart)=plot(plotfitness(restart,1:y),'--s','Color',cmap(restart,:),'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',cmap(restart,:),...
                'MarkerSize',5); %Plot the best fitness value per generation

%axis([1 y 0 max(plotfitness)+1])
xlabel('Generation')
ylabel('Fitness (Number of Mismatches)')
title('Best Fitness vs. Generation')


end


end

if javaon==1
%%%%LEGENDPLOT%%%%%%%
if y==1
plotlegend{restart}=sprintf('Run #%d',restart);
legend(hplot,plotlegend,'Location','Best');
end
%%%%%%%%%%%%%%%%%%%%%%%%%
end


end
if javaon==1
 close(k) %Closes waitbar
end

fprintf('\nMaximum Number of Generations Reached\n')
disp(population) %Displays Population after max generations are reached.
end


fprintf('\nMaximum Number of Generations and Restarts Reached\n')
Bestfit=population(1,:);
disp(population)% Displays full population after max generations and reruns are reached.
simtime=toc; %End Time

%%%%%%%%%%%%%%%%%%%%%%%%%%%%BUILD OUTPUT FILE
resultsdirectory=exist('Simulation_Results','dir');
if resultsdirectory~=7
```

```
    mkdir('Simulation_Results')
end

%Save Plots
if javaon==1
plotdirectory=exist('Plots','dir');
if plotdirectory~=7
    mkdir('Plots')
end
    plotfilename=sprintf('Plots/Plot_%g_%g_%g_%g_%g',date(1),date(2),date(3),date(4),date(5));
    saveas(gcf,plotfilename,'png')
end

filename=sprintf('Simulation_Results/Simulation_Results_%g_%g_%g_%g_%g.txt',date(1),date(2),↙
date(3),date(4),date(5));
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf('Printing to a file: %s \n',filename);
u = fopen(filename,'w'); %open output file
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf(u,'Filename - %s\n',filename);
fprintf('Simulation Completed in %g Generation(s) --- Simulation Time: %g Seconds\n',y,↙
simtime);
fprintf(u,'Simulation Completed in %g Generation(s) --- Simulation Time: %g Seconds\n',y,↙
simtime);
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm↙
\t\t Mismatches\n');
fprintf(u,'kpa \t\t kia \t\t kda \t\t tda \t\t kna \t\t kpm \t\t kim \t\t kdm \t\t tdm \t\t knm↙
\t\t Mismatches\n');
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('Best Fit\n');
fprintf(u,'Best Fit\n');
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(1,:));
fprintf↙
('=============================================================================================↙
==========================================================================\n');
fprintf↙
(u,'=============================================================================================↙
==========================================================================\n');
fprintf('Entire Population\n');
fprintf(u,'Entire Population\n');

for j=1:size(population,1)
fprintf('%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population↙
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),↙
population(j,10),population(j,11));
fprintf(u,'%-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f \t %-10.4f↙
\t %-10.4f \t %-10.4f \t %-10.4f\n',population(j,1),population(j,2),population(j,3),population↙
(j,4),population(j,5),population(j,6),population(j,7),population(j,8),population(j,9),↙
population(j,10),population(j,11));
```

```
end

fprintf↙
('==========================================================================================↙
==========================================================================\n');
fprintf↙
(u,'==========================================================================================↙
==========================================================================\n');
fclose(u);
%close output file
close_system('maurer_ap_ml_aug_07',0) % Closes Simulink System to prevent shadowing
diary off %Ends Log File
%%%%%%%%%%%%%%%END BUILD OUTPUT FILE

%End Genetic Algorithm%
end %Ends the switch function at the start
slprjdirectory=exist('slprj'); %Checks for existance of rapid accelerator folder
if slprjdirectory ~= 0
rmdir('slprj','s') % This removes the compiled rapid accelerator folder after simulation.
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%↙
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B.2 OpeningGUI.m

This file contains the code used to build the graphical user interface.

```
function varargout = OpeningGUI(varargin)
% OPENINGGUI M-file for OpeningGUI.fig
%      OPENINGGUI, by itself, creates a new OPENINGGUI or raises the existing
%      singleton*.
%
%      H = OPENINGGUI returns the handle to a new OPENINGGUI or the handle to
%      the existing singleton*.
%
%      OPENINGGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in OPENINGGUI.M with the given input arguments.
%
%      OPENINGGUI('Property','Value',...) creates a new OPENINGGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before OpeningGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to OpeningGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help OpeningGUI

% Last Modified by GUIDE v2.5 21-Nov-2010 17:08:49

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @OpeningGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @OpeningGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before OpeningGUI is made visible.
function OpeningGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to OpeningGUI (see VARARGIN)

% Choose default command line output for OpeningGUI
handles.output = hObject;

% Update handles structure
set(handles.uipanel8,'SelectionChangeFcn',@uipanel8_SelectionChangeFcn);
SearchMethod = 'radiobutton1';
assignin('base', 'SearchMethod', SearchMethod);
guidata(hObject, handles);

% Update handles structure
set(handles.uipanel9,'SelectionChangeFcn',@uipanel9_SelectionChangeFcn);
OptimizationMethod = 'genetic';
assignin('base', 'OptimizationMethod', OptimizationMethod);
guidata(hObject, handles);
```

```
% Update handles structure
set(handles.uipanel10,'SelectionChangeFcn',@uipanel10_SelectionChangeFcn);
testcondition = 'eyesopen';
assignin('base', 'testcondition', testcondition);
guidata(hObject, handles);



% UIWAIT makes OpeningGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);
clc

% --- Outputs from this function are returned to the command line.
function varargout = OpeningGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
assignin('base', 'RequestedParameters', get(handles.uitable2, 'Data'));    % Assigns the user↙
supplied parameter inputs to 'RequestedParameters' global variable.

InputUI = str2double(get(handles.InputConstraint, 'String'));
assignin('base', 'InputConstraint', InputUI);                              % Assigns the user↙
supplied 'minimum mismatches to stop at' value to 'InputConstraint' global variable.


populationsize = str2double(get(handles.populationsize, 'String'));
assignin('base', 'populationsize', populationsize );    %Assigns value to populationsize from↙
GUI

numberofgenerations = str2double(get(handles.numberofgenerations, 'String'));
assignin('base', 'numberofgenerations', numberofgenerations); %Assigns value to↙
numberofgenerations from GUI

reruns = str2double(get(handles.reruns, 'String'));
assignin('base', 'reruns', reruns );    %Assigns value to populationsize from GUI

close

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pushbutton2
DefaultParameters= [17.55;0.8;6.5;0.171;327;16.5;1.0;8.9;0.171;130];       % Manual list of↙
default parameter values listed in grey color on the left side of the 'OpeningGUI.m' GUI↙
window.
set(handles.uitable2,'Data',DefaultParameters)
```

```
% --- Executes during object deletion, before destroying properties.
function uitable1_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function InputConstraint_Callback(hObject, eventdata, handles)
% hObject    handle to InputConstraint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of InputConstraint as text
%        str2double(get(hObject,'String')) returns contents of InputConstraint as a double


% --- Executes during object creation, after setting all properties.
function InputConstraint_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputConstraint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function uipanel8_SelectionChangeFcn(hObject, eventdata)

%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
    case 'radiobutton1'
      %execute this code when fontsize08_radiobutton is selected
      SearchMethod = get(eventdata.NewValue,'Tag');
      assignin('base', 'SearchMethod', SearchMethod);

    case 'radiobutton2'
      %execute this code when fontsize12_radiobutton is selected
      SearchMethod = get(eventdata.NewValue,'Tag');
      assignin('base', 'SearchMethod', SearchMethod);

    otherwise
       % Code for when there is no match.

end
%updates the handles structure
guidata(hObject, handles);

function uipanel9_SelectionChangeFcn(hObject, eventdata)

%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
    case 'iterative'
      %execute this code when fontsize08_radiobutton is selected
      OptimizationMethod = get(eventdata.NewValue,'Tag');
      assignin('base', 'OptimizationMethod', OptimizationMethod);

    case 'genetic'
      %execute this code when fontsize12_radiobutton is selected
      OptimizationMethod = get(eventdata.NewValue,'Tag');
      assignin('base', 'OptimizationMethod', OptimizationMethod);
```

```
    otherwise
       % Code for when there is no match.

end
%updates the handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function radiobutton1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function genetic_CreateFcn(hObject, eventdata, handles)
% hObject    handle to genetic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object deletion, before destroying properties.
function genetic_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to genetic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function uipanel9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to uipanel9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes when uipanel9 is resized.
function uipanel9_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes when selected object is changed in uipanel10.
function uipanel10_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uipanel10
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
    case 'eyesopen'
      %execute this code when fontsize08_radiobutton is selected
      testcondition = get(eventdata.NewValue,'Tag');
      assignin('base', 'testcondition', testcondition);

    case 'visualfeedback'
      %execute this code when fontsize12_radiobutton is selected
      testcondition = get(eventdata.NewValue,'Tag');
      assignin('base', 'testcondition', testcondition);

    case 'eyesclosed'
      %execute this code when fontsize12_radiobutton is selected
      testcondition = get(eventdata.NewValue,'Tag');
```

```
      assignin('base', 'testcondition', testcondition);

    otherwise
       % Code for when there is no match.
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function populationsize_Callback(hObject, eventdata, handles)
% hObject    handle to populationsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of populationsize as text
%        str2double(get(hObject,'String')) returns contents of populationsize as a double


% --- Executes during object creation, after setting all properties.
function populationsize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to populationsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function numberofgenerations_Callback(hObject, eventdata, handles)
% hObject    handle to numberofgenerations (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numberofgenerations as text
%        str2double(get(hObject,'String')) returns contents of numberofgenerations as a double


% --- Executes during object creation, after setting all properties.
function numberofgenerations_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numberofgenerations (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function reruns_Callback(hObject, eventdata, handles)
% hObject    handle to reruns (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of reruns as text
%        str2double(get(hObject,'String')) returns contents of reruns as a double


% --- Executes during object creation, after setting all properties.
function reruns_CreateFcn(hObject, eventdata, handles)
% hObject    handle to reruns (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over reruns.
function reruns_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to reruns (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```